VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE, LOKALIZACE A URČENÍ PLOCHY CHRONICKÝCH RAN

DETECTION, LOCALIZATION AND DETERMINATION OF CHRONIC WOUNDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FILIP GULÁN**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Ing. MARTIN DRAHANSKÝ, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/21022/2017/xgulan00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Gulán Filip, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Detekce, lokalizace a určení plochy chronických ran**
Detection, Localization and Determination of Chronic Wounds
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s knihovnou pro zpracování obrazu OpenCV. Seznamte se s frameworkem Ionic. Prostudujte literaturu týkající se chronických ran.
2. Navrhněte aplikaci pro snímání obrazů chronických ran a následné zpracování (detekci, lokalizaci a určení plochy chronické rány).
3. Implementujte a otestujte navržený program pro zvolené prostředí (Android a Windows), vhodný pro mobilní zařízení.
4. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- Griffith Chris. *Mobile App Development with Ionic*, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova. O'Reilly & Assoc, 2017, p. 292. ISBN 1491998121.
- Bradski Gary R. *Learning OpenCV*. Sebastopol: O'Reilly, c2008. ISBN 978-0-596-51613-0.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**, UITS FIT VUT
Konzultant: Sepši Milan, MUDr., FNsP Brno
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Cieľom tejto diplomovej práce je návrh a realizácia multiplatformnej aplikácie pre detekciu, lokalizáciu a určenie plochy chronických rán. Aplikácia má za úlohu pomáhať zdravotným sestrám, doktorom a ošetrovateľom sledovať a vyhodnocovať chronické rany v priebehu trvania liečby. Aplikácia je postavená na programovacom jazyku Typescript, hybridnom aplikačnom rámci Ionic a desktopovom aplikačnom rámci Electron. Vyhodnocovanie chronickej rany prebieha na strane servera, kde sa využíva programovací jazyk Python, pre RESTful aplikačné rozhranie aplikačný rámec Flask a pre spracovanie obrazu knižnica OpenCV.

Abstract

The aim of this diploma thesis is to design and implement a multiplatform application for detection, localization and determination of the extent of chronic wounds. The application is intended to assist nurses, doctors and healthcare assistants to monitor and evaluate chronic wounds in the course of treatment. The application is based on the Typescript programming language, on the Ionic hybrid application framework and on the Electron desktop application framework. Chronic wound assessment runs on the server-side where the Python programming language is used. The Flask application framework is used for the RESTful application interface and the OpenCV library is used for image processing.

Kľúčové slová

biomedicína, chronická rana, detekcia, lokalizácia, určenie plochy, Android aplikácia, Windows aplikácia, Ionic, OpenCV, Electron

Keywords

biomedicine, chronic wound, detection, localization, determination, Android application, Windows application, Ionic, OpenCV, Electron

Citácia

GULÁN, Filip. *Detekce, lokalizace a určení plochy chronických ran.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Martin Drahanský, Ph.D.

Detekce, lokalizace a určení plochy chronických ran

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Prof.Ing.,Dipl.-Ing. Martina Drahánskeho, Ph.D. Ďalšie informácie mi poskytol pán MUDr. Milan Sepši z Fakultnej nemocnice Brno. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Filip Gulán
21. mája 2018

Poďakovanie

Týmto by som sa chcel poďakovať vedúcemu práce pánovi Prof.Ing.,Dipl.-Ing. Martinovi Drahánskemu, Ph.D. a MUDr. Milanovi Sepšimu za odborné vedenie a cenné rady, ktoré mi pomohly pri tvorbe tejto práce. Zároveň ďakujem všetkým, ktorý ma podporovali behom štúdia.

Obsah

1 Úvod **3**

2 Teoretický rozbor **4**
2.1 Koža . . . 4
2.2 Kožná rana . . . 5
2.3 Triedenie a typy rán . . . 6
2.4 Príčiny rán . . . 7
2.5 Chronické rany . . . 7
2.5.1 Bercové vredy . . . 7
2.5.2 Dekubity . . . 8
2.5.3 Diabetická noha . . . 9
2.5.4 Maligne rany . . . 10
2.5.5 Pooperačné rany . . . 11
2.6 Hojenie rán . . . 11

3 Analýza a návrh riešenia **13**
3.1 Analýza požiadaviek . . . 13
3.2 Návrh aplikácie . . . 14
3.2.1 Návrh funkcionality . . . 14
3.2.2 Návrh grafického užívateľského rozhrania . . . 15
3.3 Návrh databáze . . . 19
3.4 Návrh aplikačného rozhrania . . . 21

4 Použité technológie **22**
4.1 Klientská časť - aplikácia . . . 22
4.2 Databázová časť . . . 25
4.3 Serverová časť - aplikačné rozhranie . . . 26

5 Implementácia **28**
5.1 Implementácia klientskej aplikácie . . . 28
5.1.1 Štruktúra aplikácie a popis jej komponentov . . . 28
5.1.2 Implementačné detaily aplikácie . . . 33
5.2 Finálna forma databáze . . . 34
5.3 Implementácia serverového aplikačného rozhrania . . . 36
5.3.1 Cross-Origin Ressource Sharing . . . 37
5.3.2 Poloautomatická detekcia chronickej rany . . . 38
5.4 Implementačné detaily . . . 40
5.4.1 Prihlasovanie . . . 41

5.4.2 Generovanie PDF súborov 43
5.4.3 Získavanie snímkov 43
5.4.4 Spracovanie snímkov 44

6 Testovanie a vyhodnocovanie **46**
6.1 Testovanie serverového aplikačného rozhrania 46
6.2 Vyhodnocovanie detekcie chronickej rany 47
6.3 Ďalšie smerovanie projektu 49

7 Záver **52**

Literatúra **53**

A Obsah CD **56**

B Návod na inštaláciu **57**
B.1 Nasadenie serverového aplikačného rozhrania 57
B.2 Zostavenie aplikácie 57

C Android aplikácia **58**

Kapitola 1

Úvod

Koža je najväčší orgán ľudského teľa, ktorý pokrýva telo človeka. Kvôli tomu je prirodzené, že podlieha rôznym chorobám a je náchylná na početné zranenia. Medzi časté zranenia patria aj kožné rany, ktoré sú spôsobené rôznymi faktormi anatomického a fyziologického charakteru, či už sa jedná vnútorne príčiny, alebo vonkajšie. Rany môžu byť od tých menej vážnych, až po tie najvážnejšie. Menej vážne rany zasahujú väčšinou do pokožky, zamše a podkožného tuku. Avšak vážnejšie rany už môžu zasahovať aj nervovo-cievnych zväzkov a orgánov, a tým tak významne ohrozovať život postihnutého pacienta. Jedno z najvýznamnejších delení rán je delenie podľa priebehu. Takéto delenie rozdeľuje rany na, rany akútne a rany chronické. Akútne rany vznikajú v zdravom kožnom tkanive, hoja sa v krátkom čase a bez komplikácií. S takýmito ranami sa človek stretá často vo svojom živote, či už pri porezaní kože, odretí kože, alebo iných bežných vonkajších kožných poraneniach. Chronické rany, ktoré sú stredobodom tejto práce, na druhú stranu trvajú dlhšie než 4 týždne, a aj napriek všetkej odpovedajúcej liečbe nevykazujú tendenciu hojenia. Za najbežnejšiu príčinu takýchto rán sa považujú lokálne poruchy výživy kože, pôsobenie tlaku, alebo systémové ochorenie.

Táto diplomová práca vznikla z dôvodu neexistencie nástroja, ktorý by pomohol lekárom a zdravotným sestrám so sledovaním chronických rán, ich priebehu a vývoja v čase. Pomocou aplikácie by malo byť možné detekovať, lokalizovať a určiť plochu chronickej rany. Zároveň by malo byť možné porovnávať jednotlivé výsledky danej rany v histórií a tak adekvátne reagovať na danú situáciu zmenou liečby, poprípade zavedením opatrení proti zhoršovaniu diagnózy. V momentálnej situácií určenie plochy chronickej rany prebieha zdravotnými sestrami za pomoci pravítok a iných merných pomôcok. Táto klasická metóda získavania informácií o rane a zisťovania plochy nie je veľmi efektívna a ani presná, keďže rany môžu mať rôzne tvary a podoby.

Nasledujúca kapitola 2 sa venuje teoretickému rozboru a obsahuje informácie potrebné k základnému pochopeniu problematiky chronických rán. Ďalšia kapitola 3 sa zaoberá analýzou a návrhom samotnej mobilnej aplikácie a jej serverových súčastí. Konkrétne popisuje návrh grafického užívateľského rozhrania, ale aj návrh aplikačného rozhrania REST a výber vhodných technologických prostriedkov. Hneď potom nasleduje kapitola 4, ktorá rozoberá technológie, ktoré boli potrebné pri vývoji aplikácie a jej súčastí. V poradí ďalšou a určite najvýznamnejšou kapitolou je kapitola 5 pojednávajúca o samotnej implementácií. Celá práca je ukončená kapitolou 6 o testovaní a vyhodnocovaní implementovaného riešenia a sú navrhnuté ďalšie kroky a smerovanie práce.

Kapitola 2

Teoretický rozbor

Táto kapitola sa zaoberá teoretickým rozborom problematiky chronických rán, ktoré sú jadrom tejto práce. Kapitola sa snaží najprv vysvetliť základné pojmy a termíny týkajúce sa problematiky a následne vysvetľuje samotné chronické rany, ich príčiny vzniku a prejavy. Celá kapitola vychádza z informácií dostupných na webovej stránke [11] a z kníh [31] a [38].

2.1 Koža

Koža je najväčší orgán ľudského tela[11], ktorý má u dospelého človeka veľkosť plochy od 1,5 až do 2 metrov štvorcových[38] a váži približne 12-16 percent celkovej telesnej hmotnosti[38]. Hrúbka kože sa pohybuje od 0,5 až po 4 milimetre a líši sa podľa jednotlivých častí tela[31]. Najtenšia je na očných viečkach, najhrubšia zase na chrbte [11].

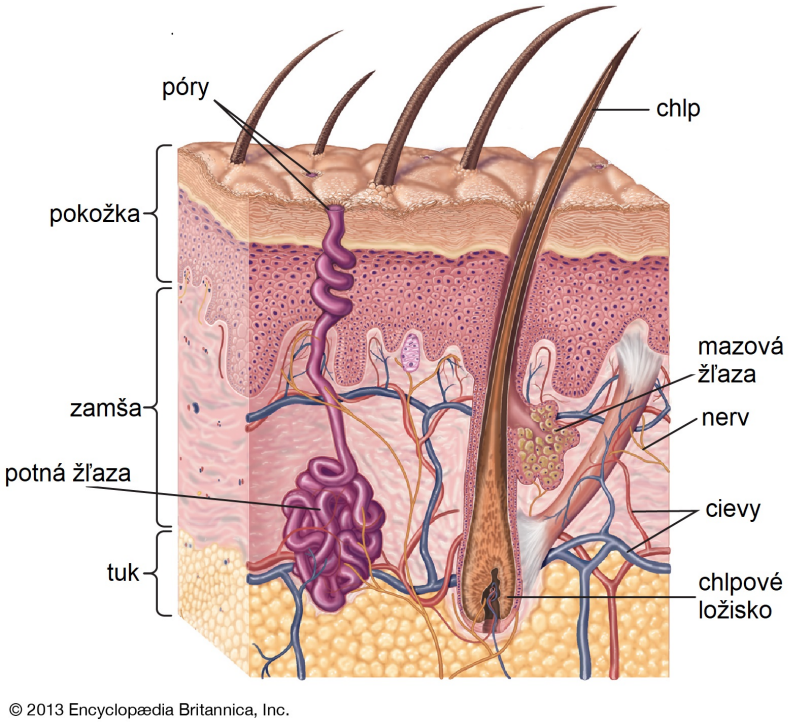
Kože nielenže plní funkciu obalu človeka, ale okrem toho plní aj celú radu pre človeka oveľa dôležitejších funkcií, medzi ktoré patrí [38]:

- **ochrana -** pred vstupom nečistôt do organizmu, pred stratou tekutín, proti teplotným výkyvom
- **termoregulácia -** tepelná izolácia, výmena tepla medzi organizmom a okolím
- **vnímanie -** nervové zakončenia v koži informujú o teplote alebo poranení
- **skladovanie a vstrebávanie -** koža sa podiela na látkovej výmene a uchovávaní tukov, vody, minerálnych látok a vitamínov a prenikajú ňou do organizmu látky rozpustné v tukoch a dýchacie plyny
- **vylučovanie -** mazové a potné žľazy vylučujú vodu, soli, tuky, oxid uhličitý a dusíkaté látky
- **metabolická funkcia -** koža je významný metabolický orgán, ktorý syntetizuje melanin, vitamín D a podieľa sa pri tvorbe protilátok
- **estetická a komunikatívna funkcia -** vzhľad a úprava kože je jedným z prvých znakov, ktorých si ľudia pri vzájomnom kontakte všímajú. Stav kože súvisí so sebahodnotením a sociálnou adaptabilitou

Kôža je zložená z 3 hlavných časti, a to z pokožky (epidermis), zamši (dermis) a podkožia (subcutis). Tieto 3 hlavné vrstvy je možné vidieť na obrázku 2.1 dole. Okrem týchto

hlavných vrstiev sa skladá aj z vlasov/chlpov, nechtov a žliaz (mazové, potné, pachové a mliečne). Dokopy sa nazývajú kožné deriváty [38].

- **Pokožka -** vonkajšia vrstva. Je tvorená niekoľkými vrstvami kožných buniek. V spodných častiach pokožky sa bunky neustále delia a vytláčajú bunky nad sebou bližšie k povrchu. Postupom do horných vrstiev pokožky bunky postupne rohovatejú, odumierajú a odlupujú sa. Pomocou tohoto procesu dochádza k plynulej obmene celej pokožky. Z človeka sa počas jeho života odlúpne v priemere asi 20 kilogramov mŕtvych buniek [11].
- **Zamša -** stredná vrstva. Je tvorená väzivom, sieťou ciev a nervových zakončení. Táto vrstva rozhoduje o pevnosti kože, jej pružnosti a mechanickej odolnosti. Medzi dôležité súčasti zamše patria hlavne nervové zakončenia, vďaka ktorým vnímame teplo, chlad a bolesť. Jemné cievy zase slúžia pre reguláciu tepla a imunitné bunky zaisťujú ochranu [11].
- **Podkožie -** najhlbšia vrstva. Je tvorená riedkym väzivom a tukom. Hlavnou úlohou podkožia je ochrana proti teplotným vplyvom a mechanickému poškodeniu. V tukovom tkanive si organizmus uchováva prebytky energie [11, 31].



Obr. 2.1: Zloženie ľudskej kože. Modifikované zo zdroja [27]

2.2 Kožná rana

Kožná rana je definovaná ako narušenie kontinuity kožného povrchu a integrity organizmu, narušenie anatomickej štruktúry a funkcie kože spôsobené rôznymi príčinami zasahujúcimi rôzne hlboko do podkožných tkanív [31].

Kožné rany je možné rozdeliť na jednoduché a komplikované. Jednoduché kožné rany zasahujú do pokožky, zamše a podkožného tuku. Komplikované rany narozdiel od jednoduchých prenikajú hlbšie a poškodzujú dôležité nervovo-cievne zväzky a orgány. Komplikované rany je ďalej možné rozdeliť na penetrujúce a nepenetrujúce. Penetrujúca rana preniká do teľnej dutiny. Nepenetrujúca rana naopak nepreniká do telesnej dutiny. U každej rany, či už jednoduchej, komplikovanej, penetrujúcej, alebo nepenetrujúcej určujeme a popisujeme vlastnosti, ktoré sú veľmi dôležité pri sledovaní procesu hojenia rany a voľby optimálnej liečby. Medzi takéto vlastnosti patrí [11]:

- lokalizácia rany
- veľkosť rany
- hĺbka rany
- tvar rany
- smer rany
- okraje rany

2.3 Triedenie a typy rán

Rany a poškodenia je možné deliť do niekoľko skupín podľa rôznych kritérií. Rany sa delia podľa ich priebehu, podľa rozsahu ich poškodenia, podľa množstva choroboplodných zárodkov, podľa spôsobu hojenia, podľa lokalizácie, podľa postihnutých štruktúr a mnoho iných delení, pričom najdôležitejšie delenia sú tie, ktoré sú ďalej popísané podrobnejšie [31].

Rany podľa priebehu sa delia na:

- **akútne rany -** vznikajú v zdravom kožnom tkanive. Hoja sa obvykle v krátkom čase a bez komplikácií.
- **chronické rany -** trvajú dlhšie než 4 týždne, alebo to sú rany, ktoré vznikajú v zmenenom tkanive, a ktoré aj cez všetku odpovedajúcu liečbu nevykazujú známky hojenia.

Rany podľa rozsahu sa delia na:

- **plošné rany -** odtrhnutie kože s podkožím
- **povrchové rany -** poškodenia kožného krytu, alebo časti podkožia
- **hlboké rany -** poškodenie až do podkožia

Rany podľa množstva choroboplodných zárodkov sa delia na:

- **čisté rany -** neinfikované rany bez zárodkov a bez príznakov zápalu (chirurgický rez)
- **čisté kontaminované rany -** rany bez dôkazu infekcie vzniknuté pri operácií orgánov dýchacieho, tráviaceho, pohlavného alebo močového ústrojenstva
- **kontaminované rany -** penetrujúce traumatické rany, pri ktorých sú prítomné príznaky zápalu

- **infikované rany -** rany obsahujúce odumreté tkanivo s premnoženými mikroorgnizmami (rany vzniknuté po pohryznutí, poprípade zanedbané, alebo zastaralé rany)

Rany podľa spôsobu hojenia sa delia na:

- **rany s primárnym hojením -** vďaka zlepeniu/zošitiu okrajov rany nevzniká nové spojivové tkanivo
- **rany so sekundárnym hojením -** rana sa hojí novo tvoreným tkanivom

2.4 Príčiny rán

Rany môžu vznikať pôsobením rôznych príčin, medzi ktoré patria vonkajšie, vnútorné príčiny alebo kombinácia týchto dvoch. Medzi vonkajšie príčiny patria rany rezné, sečné, tržné, rany spôsobené pohryznutím, bodné, strelné, zhmoždené, popáleniny, omrzliny, poleptaniny a rany spôsobené z ožiarenia. Medzi vnútorné príčiny patria cievne vredy dolných končatín, neuropatické vredy, preležaniny, rany pri nádorových ochoreniach, rany pri infekčných ochoreniach a rany pri imunitných poruchách. Vo všeobecnosti sa dajú príčiny vzniku rán rozdeliť do troch základných oblastí a to podľa toho, na akom základe vznikajú [31]:

- lokálne poruchy výživy kože
- lokálneho pôsobenia tlaku, cievneho poškodenia
- systémového ochorenia (infekčného, nádorového, krvného...)

2.5 Chronické rany

V kapitole Triedenie a typy rán boli rany rozdelené podľa priebehu na akútne a chronické.

Akútne rany nie sú predmetom tejto práce, avšak pre celistvosť sú tu uvedené. Akútna rana je porušenie integrity tkaniva teľa vzniknuté v dôsledku fyzikálneho, mechanického, alebo termického poškodenia. Vznikajú v zdravom kožnom tkanive. Hoja sa obvykle v krátkom čase a bez komplikácií. Ich príčinou je väčšinou úraz, alebo chirurgický zákrok. Patria sem rany mechanické, traumatické, termické, chemické, aktinické, opary a pľuzgiere [11].

Chronická rana je sekundárne hojaca sa rana, ktorá sa musí uzavrieť pomocou výstavby nového tkaniva. Vzniká najčastejšie následkom lokálnych porúch vyživovania kože podmienených cievnym, alebo neurologickým ochorením, prípadne dlhodobým lokálnym pôsobením tlaku. Aj napriek adekvátnej liečbe sa takáto rana nezahojí skôr ako za 9 týždňov. Doba hojenia je spravidla dlhá a závisí na príčine a rozsahu poškodeného tkaniva. Chronické rany môžu vzniknúť aj z akútnych rán a to tak, že akútna rana je neadekvátne ošetrená, prípadne vznikne infekcia. Chronické rany sa taktiež objavujú v patologicky zmenených tkanivách. Medzi najčastejšie chronické rany patria bercové vredy, dekubity (preležaniny), diabetická noha, nádory s vredovým rozpadom, alebo komplikovane sa hojace pooperačné rany [31].

2.5.1 Bercové vredy

Bercové vredy sú poškodením kožného krytu zasahujúce rôzne hlboko do podkožného tkaniva v oblasti členku. Nachádza sa najčastejšie v miestach medzi kolenom a členkom, hlavne v oblasti vnútorného členku.

Príčinou vzniku vredov býva drobné poranenie, alebo infekcia. Väčšina bercových vredov je žilného pôvodu a sú prejavom hromadenia krvi v dolných končatinách. Pri pretlaku v žilnom riečisku dochádza k žilnej nedostatočnosti. Žily na končatinách sa rozširujú, chlopne strácajú svoju správnu funkciu a prepúšťajú časť krvi naspäť. Prietok končatinami sa spomaľuje, tekutina z krvi v preplnených žilách prestupuje do podkožia, rozvíja sa opuch, ktorého následkom dochádza k poruche výživy kože.

Prvým prejavom toho, že vzniká bercov vred je miestne začervenanie. Koža nad postihnutým miestom sa ztenčuje, je suchá, môže ju prekrývať šedivý ekzém a toto miesto je často dobre ohraničené s rovnými okrajmi. Bercove vredy bývajú rozsiahle, vlhké, skôr plytké a s povlečenou spodinou. Väčšinou sú sprevádzané rozsiahlym opuchom postihnutej oblasti a ďalšími kožnými zmenami. Okolie vredu vykazuje známky žilnej nedostatočnosti, hromadenie pigmentu, stráta ochlpenia. Bercové vredy sú často bolestivé [11, 38].

Obr. 2.2: Bercov vred žilného pôvodu [6]

2.5.2 Dekubity

Dekubity, známe tiež ako preležaniny, sú rany spôsobené pôsobením lokálneho tlaku na kožu. Vznikajú v miestach dlhodobého kontaktu kože s podložkou. Sú typické pre chorých ľudí, ktorí sú dlhodobo pripútaní na lôžko.

Príčinou vzniku preležením je to, že v miestach neustáleho kontaktu a tlaku dochádza k uzatvoreniu drobných ciev. Tkanivo je zle zásobované živinami a kyslíkom, a tak dochádza k ich postupnému odumieraniu. Rozsah odumretia tkaniva závisí na vzájomnom pôsobení niekoľkých faktorov. Konkrétne závisí na intenzite tlaku, doby pôsobenia tlaku, odolnosti organizmu voči tlaku, celkového stavu postihnutia a vplyvov vonkajšieho prostredia. Preležaniny sa objavujú pomerne rýchlo, v ojedinelých prípadoch už po niekoľkých hodinách.

K miestam najnáchylnejším k vzniku dekubitov patria oblasti s malou vrstvou tukového a svalového tkaniva, kde z vonka pôsobí tlak priamo na kosti. Medzi takéto oblasti patrí oblasť nad krížovou kosťou, päty, sedacie kosti, oblasť nad veľkými výčnelkami stehennej kosti a vonkajšie členky. Avšak napriek tomu sa môžu vytvoriť kdekoľvek na tele. Vývoj preležanín prebieha v niekoľkých štádiách. Prvým prejavom je začervenanie, bolestivosť a

opuchnutie kože. Nasleduje pľuzgier, alebo povrchový vred zasahujúci do pokožky a zamše, v okolí sa rozvíja zápal. Posledným štádiom je odumretie tkaniva. Nedostatočne liečená preležanina sa ďalej prehlbuje, odumreté tkanivo sa topí, hnilobne páchne a zvyšky tkaniva majú žltozelenú farbu [11, 31].

Obr. 2.3: Preležanina v druhom štádiu [42]

2.5.3 Diabetická noha

Syndrómom diabetickej nohy sa označujú defekty dolných končatín, spôsobené postihnutím ciev a nervov. Ide o komplikáciu ochorenia cukrovky.

Medzi 2 hlavné príčiny vzniku tohoto syndrómu sa dá označiť diabetická neuropatia a ischemická choroba dolných končatín. Diabetickou neuropatiou je myslená dlhodobo zvýšená hladina cukru v krvi, ktorá poškodzuje funkciu a štruktúru periférnych nervov. Postihnutie sa týka všetkých typov nervových vlákien. Obmedzenie, alebo strata funkcie ma priamy dopad na nohu:

- **senzorická neuropatia -** vedie k strate citlivosti na bolesť, teplotu, vibrácie, tlak a polohocit.
- **motorická neuropatia -** vedie k oslabeniu a skráteniu svalstva a vzniku otlakov.
- **vegetatívna neuropatia -** vedie k zníženému poteniu kože, strate jej pružnosti a tým k vyššiemu riziku prasklín.

Za ischemickú chorobu dolných končatín sa označuje ochorenie tepien, ktoré spôsobuje nedostatočné prekrvenie dolných končatín. Hlavnou príčinou ochorenia je ateroskleróza, ktorá má za následok zúženie, alebo úplný uzáver tepien a obmedzenie prítoku krvi. Okrem veľkých ciev sú v prípade diabetickej nohy postihnuté aj jemné vlásočnice. Takto nedostatočne prekrvené tkanivo je náchylné k poraneniu a vykazuje zlú hojivosť.

Medzi prejavy diabetickej nohy v kľude patria nepríjemné pálčivé bolesti, brnenie, alebo mravenčenie končatín. Pri chôdzi sa zase prejavuje pocitom stiahnutia okolo členkov. Koža nohy je suchá a šúpe sa. Svaly lýtok sú oslabené, ochlpenie je preriedené, alebo chýba úplne. Pri nedokrvení končatín sa môže objaviť aj bolesť, ale nie je to pravidlom. Najčastejším miestom vzniku sú členky, priehlavok, päta a prsty. Medzi ďalšie prejavy patria praskliny, pľuzgiere a odreniny, ktoré keď sa neliečia, tak sa rozširujú, poprípade sa do toho pridá infekcia [11, 38].

Obr. 2.4: Syndróm diabetickej nohy [8]

2.5.4 Maligne rany

Nehojace sa rany a kožné defekty, ktoré vznikajú pri nádorových ochoreniach sa nazývajú maligne rany. Vyskytujú sa najčastejšie u pacientov s pokročilými nádorovými ochoreniami. Vznikajú v súvislosti so spinocelulárnym karcinonom, bazocelulárnym karcinonom, malignym melanonom, alebo zhubným nádorom (pŕs, hlavy, krku,...).

Príčiny vzniku týchto rán sa dajú zhrnúť do 3 oblastí: Primárne kožný nádor, prerastanie nádoru mäkkých tkanív do kože a kožné metastázy nádoru. Rastom nádoru dochádza k narušeniu kožných kapilár a lymfatických ciev a k útlaku okolných tkanív. To vedie k obmedzeniu prísunu živín a vzniku nekrózy.

Na začiatku sa v koži najčastejšie objaví zatvrdnutie, ktoré je nasledované začervenaním a rozvojom nekrózy. Maligne rany doprevádza prevažne krvácanie, sekrécia rany, nepríjemný zápach, bolesť a svrbenie okolia rany [11, 31].

Obr. 2.5: Malígny melanóm [45]

2.5.5 Pooperačné rany

Už ako samotný názov napovedá, ide o rany, ktoré vznikli v rámci operačného výkonu. Spôsob a vzhľad obnovenia integrity kože nerušenej pri operačnom výkone záleží na chirurgovi, ktorý zákrok vykonával. O tom, akým spôsobom sa ale bude rana hojiť, a ako bude vypadať výsledná jazva, rozhoduje celá rada ďalších faktorov.

Príčinou komplikovaného hojenia operačnej rany je väčšinou infekcia. Kontaminácia rany mikroorganizmami môže byť spôsobená samotným charakterom operácie, avšak ale aj porušením aseptických pravidiel, alebo následným preväzom. Väčšina ranných infekcií je avšak spôsobená mikroorganizmami, ktoré boli v organizme prítomné už pred vznikom infekcie. K najčastejším pôvodcom patria stafylokoky, streptokoky, klebsiely, kvasiny, alebo pseudomonády.

Prejavom takýchto rán je v rannom štádiu opuch a začervenanie rany a jej okolia. Ide o prejav zápalu, ktorým sa organizmus bráni pred poškodením. Tieto prejavy sú v rôznej miere zastúpené pri všetkých operačných ranách. Až prítomnosť hnisu je známkou infekcie. Infekcia sa prajavuje tak, že žltá, alebo nazelenalá tekutina sa hromadí v rane a zväčšuje jej objem, alebo dokonca vyteká. Rana sa potom následne rozpadá, jej stehy sú povlečené a priľahlé rany nekrotické. Infekcia zasahuje do rôznej hĺbky. V niektorých prípadoch sa obmedzuje iba na kožu a podkožie, v iných zase postihuje všetky vrstvy rany [11].

Obr. 2.6: Operačná rana po klasickej laparotómii [15]

2.6 Hojenie rán

Hojenie je fyziologický proces, pri ktorom dochádza k obnove porušenej štruktúry a funkcie kože. Je to reparačný proces, pri ktorom je poškodené tkanivo nahradené väzivovým tkanivom, ktoré sa postupne mení na jazvu. Hojenie rany ovplyvňuje niekoľko faktorov, ktoré sa dajú rozdeliť na lokálne a celkové. Medzi lokálne faktory patrí porucha krvného zásobovania, stav okolitého tkaniva, pôsobenie tlaku, prítomnosť infekcie, nevhodné šicie materiály a technika, pohyb v rane, teplota, pH, dehydratácia a opuch. Medzi celkové

faktory ovplyvňujúce hojenie patrí vek, celkový zdravotný stav pacienta, stav imunitného systému, anémia, strata krvi, podvýživa, nedostatok bielkovín, vplyv liekov, imobilita a psychický stav [11, 38].

Hojenie rán prebieha v niekoľkých vzájomne prelínajúcich sa, časovo prekrývajúcich sa a na seba nadväzujúcih fází a prebieha odlišne pri akútnych ranách a chronických ranách. Hojenie chronických rán je o niečo komplikovanejšie než hojenie akútnych rán. Tieto rany sa zaceľujú výstavbou nového tkaniva. Doba ich hojenia je veľmi zdĺhavá a závisí na rozsahu poškodenia tkaniva. Hojenie prebieha v 3 hlavných fázach [38]:

1. **fáza zápalu -** je charakteristická snahou odstrániť všetky nežiaduce zložky, zaistiť odlúčenie poškodených a odumretých tkanív. Liečba je zameraná na podporu samočistiacich procesov v kombinácií spolu s chirurgickým ošetrením.

2. **fáza granulácie -** po vyčistení z fáze 1 sú vytvorené podmienky pre rast a delenie nových buniek a vzniká granulačné tkanivo, ktoré je podkladom pre proces epitelizácie.

3. **fáza epitelizácie -** finálna etapa, v ktorej dochádza k deleniu a pohybu kožných buniek. Z okrajov rany prerastá epitel a pokrýva granulačné tkanivo novo-vytvorenou kožou.

Hodnotenie hojenia rany prebieha podľa niekoľkých základných vlastností. Medzi tieto vlastnosti patrí príčina vzniku, vek rany, veľkosť, okraje, lokalizácia, hĺbka, vzhľad spodiny, bolestivosť, prítomnosť infekcie, množstvo sekrécie, zápach, okolie rany a súčasná a minulá lokálna terapia. Informácie boli prebrané z [11].

Kapitola 3

Analýza a návrh riešenia

V tejto kapitole je popísaný postup analýzy a návrhu riešenia tejto diplomovej práce. Najprv je popísaná analýza požiadaviek na výslednú aplikáciu a jej primárnu funkcionalitu. Ďalej je popísaný návrh samotnej mobilnej aplikácie, kde sú predstavené prípady užitia a použitia aplikácie, ako aj základný návrh grafického užívateľského rozhrania. Potom nasleduje návrh databáze, ktorý bol realizovaný pomocou ER diagramu. Posledná kapitola pojednáva o návrhu aplikačného rozhrania vychádzajúceho z princípov RESTu.

3.1 Analýza požiadaviek

Analýza požiadaviek čiastočne vychádza z kapitoly 2, v ktorej boli popísané dôležité teoretické znalosti a zároveň vychádza z informácií poskytnutých Fakultnou nemocnicou Brno a jej požiadavok. Podľa týchto informácií a požiadavok by malo ísť o aplikáciu určenú pre platformy Android, iOS a Windows Mobile, ktorá by mala umožňovať vyfotografovať chronickú ranu, primárne preležaninu, spolu s nejakou mierkou. Mierka by mala byť buď pero, alebo prst zdravotnej sestry, prípadne ošetrujúceho lekára a mal by byť nastaviteľný jeho rozmer. Aplikácia by mala byť schopná následne odfotografovanú ranu detekovať, lokalizovať a spočítať plošný rozmer cieľovej plochy, napríklad v milimetroch štvorcových alebo inej miere. V aplikácií by mala byť možnosť prihlásenia užívateľa pomocou sociálnych sietí, ako napríklad Facebook alebo Google+. Ďalej by mal mať užívateľ možnosť svoje snímky pomenovať, porovnať s minulými snímkami, mať možnosť tlače týchto snímok a prípadne ich ukladať na vzdialené úložisko, aby k nim mal prístup aj z iného zariadenia ako je aktuálne používané. Aplikácia by mala byť primárne cielená na zdravotné sestry, poprípade ošetrovateľov starajúcich sa o chronické rany a pomáhať im so sledovaním vývoja rán v čase.

Z vyššie uvedeného vyplýva, že aplikácia by mala byť dostupná na väčšinu platforiem, aj keď nie v rámci tejto diplomovej práce, ale predovšetkým v rámci dlhšieho časového horizontu vývoja. Z tohoto dôvody bolo usúdené, že by bolo dobré, ak by sa vytvorila hybridná aplikácia, ktorú stačí naprogramovať raz a už iba s drobnými úpravami je ju možné potom publikovať na väčšine dostupných platforiem. Keďže samotné spracovanie obrazu má v hybridných aplikáciách, webových aplikáciách a celkovo v Javasciptovom svete mizivú podporu, tak by spracovanie obrazu (konkrétne detekcia, lokalizácia a určenie plochy) nastávalo vzdialene na servery. Kvôli tomu bolo rozhodnuté, že programová realizácia by mala byť rozdelená na niekoľko logických celkov, ktorých existencia by bola navzájom od seba nezávislá. Mala by sa skladať zo samotnej mobilnej aplikácie, ktorá môže byť chápaná aj

ako klientská časť realizácie, keďže ju bude vidieť každý používateľ aplikácie. Ďalej by išlo o databázovú časť, do ktorej by sa ukladali samotné dáta a potrebné informácie spojené s jednotlivými ranami, užívateľmi a snímkami. A nakoniec ako spojovník medzi týmito 2 časťami by figurovalo aplikačné rozhranie vzdialené na serverovej časti, ktoré by komunikovalo s klientskými aplikáciami, ukladalo a získavalo dáta z databáze, umožňovalo autorizáciu a autentifikáciu používateľov, a spracovávalo snímky rán.

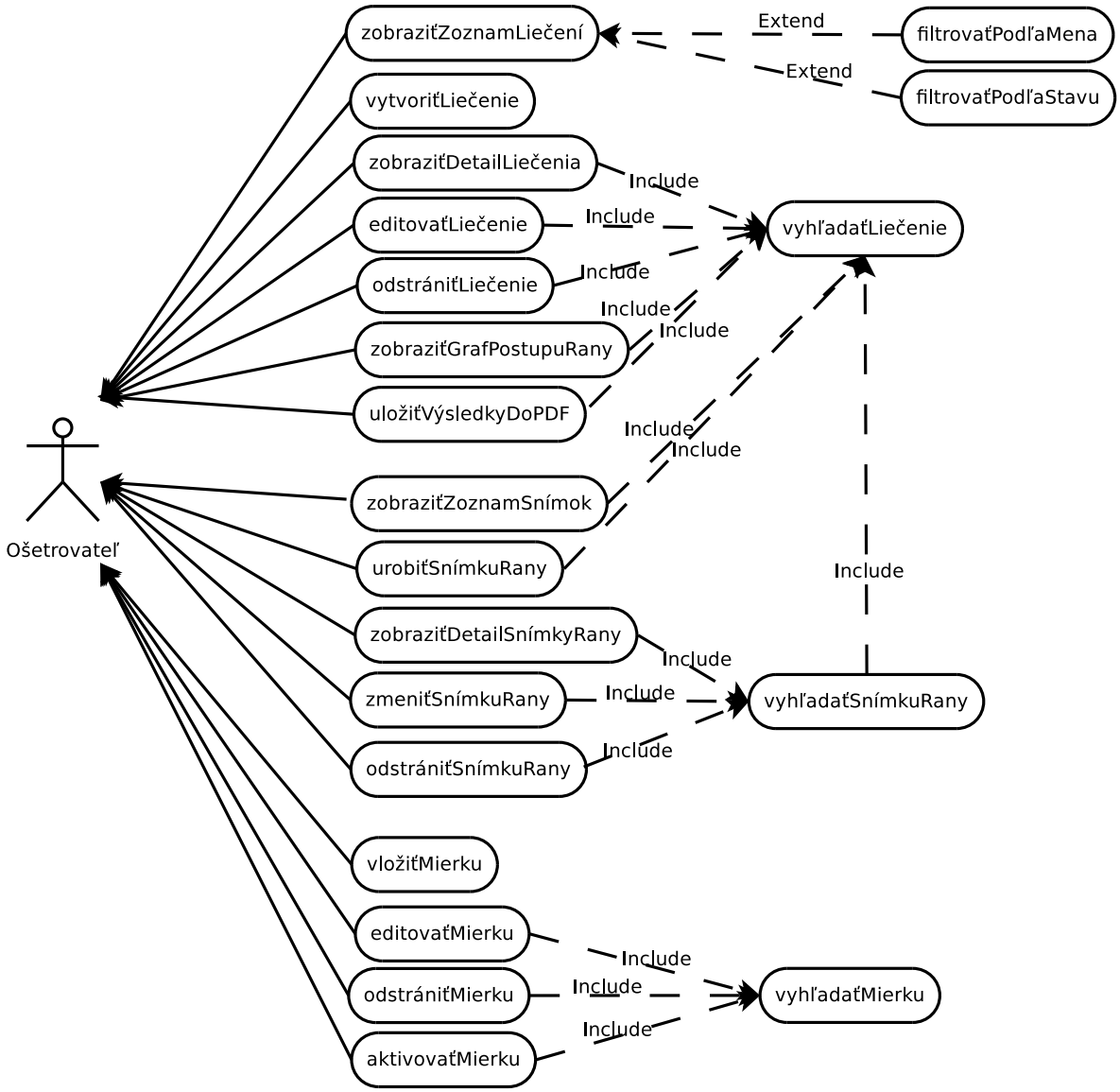
3.2 Návrh aplikácie

Návrh samotnej aplikácie bol vypracovaný s rešpektom k analýze požiadaviek. Bolo nutné premyslieť, čoho všetkého by mala byť daná aplikácia schopná, ako by mala plniť svoju úlohu čo najlepšie a čo by mal byť jej hlavný cieľ okrem samotnej detekcie, lokalizácie a určenia plochy chronickej rany. Aplikácia by nemala byť robustným informačným systémom, skôr iba čo možno najjednoduchšie poňatou aplikáciou, zbytočne nezaťažujúcou užívateľa, ale ponúkajúcou dostatočné možnosti pre pomoc so sledovaním chronických rán. Samotný návrh aplikácie bol rozdelený na 2 podkapitoly, a to na návrh funkcionality, alebo toho, čo by mal byť užívateľ schopný vykonávať, a do návrhu užívateľského rozhrania.

3.2.1 Návrh funkcionality

Návrh funkcionality aplikácie bol spracovaný pomocou Diagramu prípadov užitia (Use Case Diagramu), ďalej iba UCD. UCD zobrazuje chovanie systému z pohľadu užívateľa. Jeho účelom je popísať funkcionalitu systému, presnejšie čo presne sa od samotného systému očakáva. Diagram zobrazuje to, čoho má systém, v tomto prípade aplikácia, byť schopný ale už nehovorí o tom, ako sa to bude vykonávať. Diagram je zložený z prípadov užitia (Use Case), aktérov (Actor) a vzťahov medzi nimi. Prípad užitia je sada niekoľkých akcií, ktoré vedú k dosiahnutiu určitého cieľa. Aktér je rola, ktorá komunikuje s jednotlivými prípadmi užitia. UCD je teda v podstate diagram, ktorý zobrazuje všetko to, čo daná užívateľská rola, alebo konkrétny užívateľ môže v systéme vykonávať.

Jedinou rolou v aplikácií, tvorenej v rámci tejto diplomovej práce, je rola všeobecného užívateľa, ktorým je väčšinou zdravotná sestra, lekár, alebo iný ošetrovateľ starajúci sa o chronickú ranu. Každý užívateľ, ktorý si zapne aplikáciou, by sa mal automaticky stať roľou ošetrovateľa. Ošetrovateľ by mal mať možnosť zobraziť si zoznam liečení. Tento zoznam by mal obsahovať liečenia, ktoré aktuálne prebiehajú a sú v určitom štádií, ale zároveň aj liečenia, ktoré už nie sú aktívne pozorované a považujú sa za ukončené. Tento zoznam by mal byť filtrovateľný minimálne podľa stavu liečby a malo by sa v ňom zároveň dať vyhľadávať podľa mena liečeného pacienta. Ďalej by mal byť ošetrovateľ schopný vytvoriť nové liečenie, zobraziť detail už existujúceho liečenia a mať možnosť meniť toto liečenie. Okrem toho by mal mať možnosť zobraziť graf postupu liečenia, konkrétne graf zmeny veľkosti detekovanej plochy chronickej rany a mať možnosť tieto výsledky uložiť do PDF formátu, poprípade vytlačiť. Pre každé liečenie by mal mať možnosť ošetrovateľ potom zobraziť zoznám vytvorených snímkov rán, urobiť novú snímku, zobraziť detail existujúcej snímky, zmeniť snímku a prípadne ju odstrániť. Dôležitým aspektom v aplikácií by mala byť mierka, ktorej význam bude vysvetlený v ďalších kapitolách. Užívateľ aplikácie by mal mať možnosť zobraziť existujúce mierky, tak ako ich aj meniť a odstraňovať. Mierky by sa mali dať tiež pridávať a malo by sa dať voliť, ktorá bude nastavená ako východzia. Celý Diagram prípadov užitia je možné vidieť na obrázku 3.1.

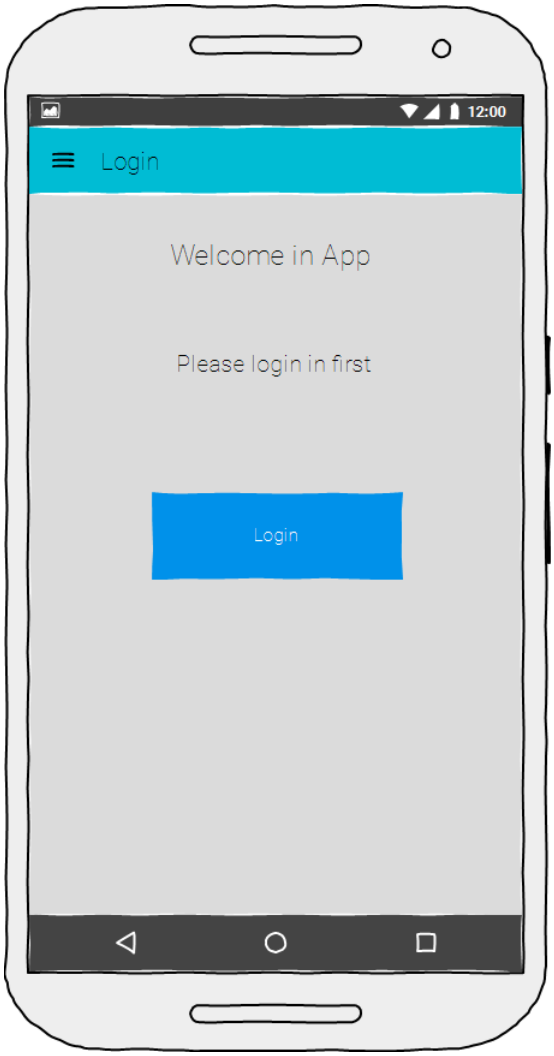


Obr. 3.1: Diagram prípadov užitia užívateľa aplikácie

3.2.2 Návrh grafického užívateľského rozhrania

Grafické užívateľské rozhranie bolo navrhnuté najprv na papier a potom pre lepšie predstavenie boli tieto náčrty prevedené do digitálnej podoby programom Wireframe Sketcher.

Prvá obrazovka, ktorá bola navrhnutá je obrazovka prihlásenia. Na túto obrazovku by sa mal dostať užívateľ vtedy, keď nie je prihlásený. Typicky sa tak deje pri prvom spustení aplikácie. Pri ďalších spusteniach by si aplikácia mala zapamätať užívateľa a prihlasovať ho už automaticky. Prihlásenie do aplikácie by malo byť povinné a v prípade, že nie je užívateľ prihlásený tak by sa nemalo dať pokračovať ďalej. Na tejto obrazovke by sa malo nachádzať iba tlačidlo na prihlásenie cez nejakú sociálnu sieť, či už Facebook, Google+ alebo obidvoje. Okrem samotného tlačidla by mal byť prítomný aj nejaký vysvetľujúci text. Návrh prihlasovacej obrazovky je možné vidieť na obrázku 3.2.

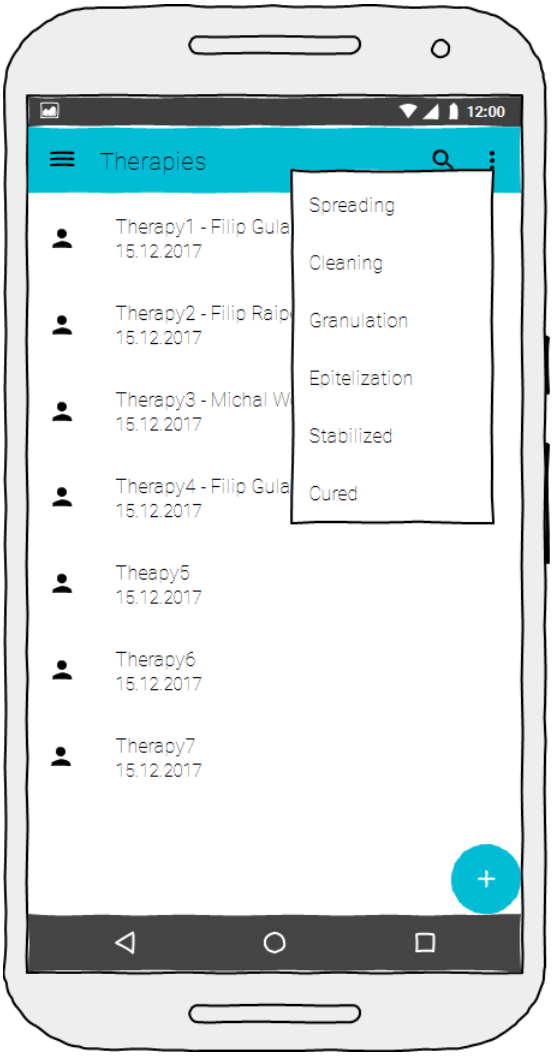


Obr. 3.2: Obrazovka prihlásenia

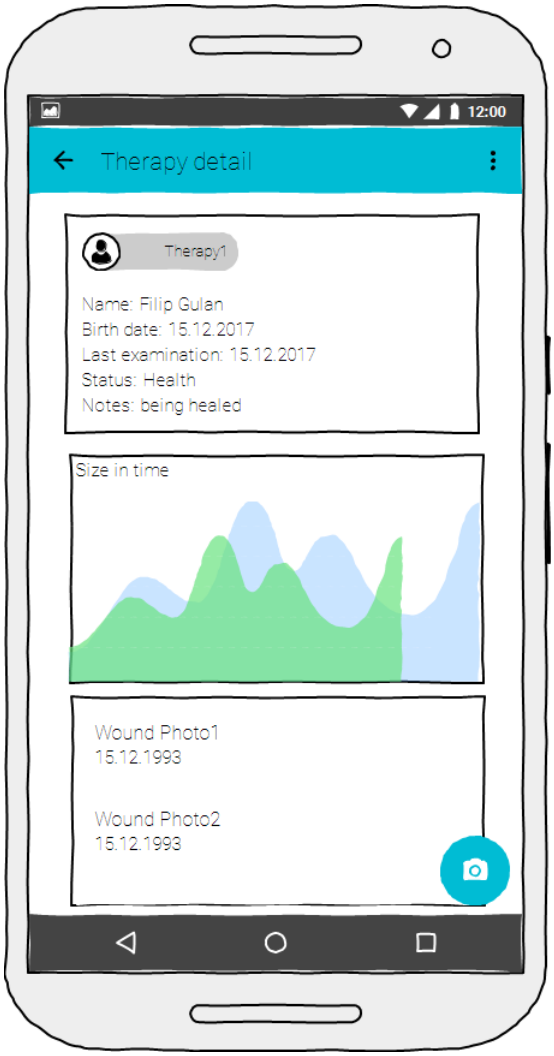
Po prihlásení do systému aplikácie by mal mať užívateľ zobrazený zoznam všetkých liečení, mala by tam existovať možnosť filtrácie a vyhľadávania umiestneného na panely aplikácie a vo vyskakovacom menu. Každý prvok zoznamu by mal mať zobrazené svoje meno liečenia, meno liečeného pacienta a posledný dátum kontroly (lepšie povedané posledný dátum fotografovania rany). V pravom dolnom rohy by malo byť plávajúce tlačidlo na pridávanie liečení. Návrh je možné vidieť na obrázku 3.3.

Po výbere konkrétnej položky by mal byť užívateľ prenesený na obrazovku detailu liečby. V tejto obrazovke by mal mať možnosť vidieť všetky informácie o liečbe (ako napríklad meno pacienta, narodenie pacienta, stav liečby, stav rany...), graf znázorňujúci vývoj rany v čase a zoznam odfotografovaných rán obsahujúci ich mená a dátumy zachytenia snímky. V pravom dolnom rohy by malo byť plávajúce tlačidlo na odfotografovanie rany. Liečbu by bolo možné na tejto obrazovke editovať, poprípade vymazať výberom z vyskakovacieho menu umiestneného v pravom hornom rohu. Mockup je znázornený na obrázku 3.4.

V prípade, že sa užívateľ rozhodne fotiť ranu, tak by sa mala aktivovať aplikácia kamera, pomocou ktorej by urobil fotku a táto fotografia by bola následne prenesená na server pre



Obr. 3.3: Obrazovka zoznamu liečení

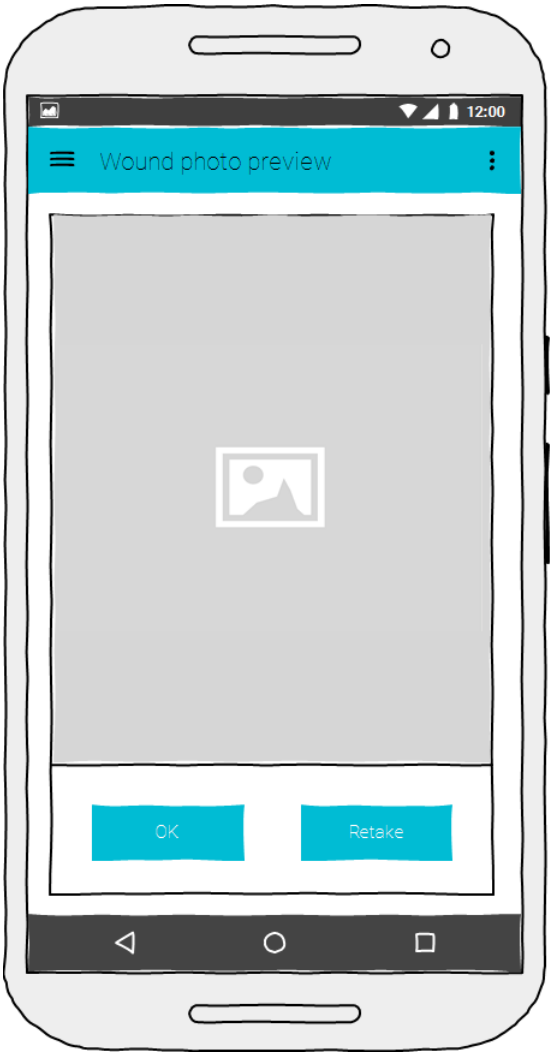


Obr. 3.4: Obrazovka detailu liečenia

detekciu, lokalizáciu a určenie plochy rany. Po tejto detekcií by bola užívateľovi vrátená upravená fotka zo serveru obsahujúca ohraničenie rany a bola by mu zobrazená ako náhľad, podľa ktorého by sa rozhodol, či mu vyhovuje a chce ju zachovať, alebo či chce urobiť vhodnejší snímok rany. Táto obrazovka náhľadu nasnímanej, detekovanej a lokalizovanej rany je možné vidieť na obrázku 3.5.

V prípade, že sa užívateľ rozhodne, že mu odfotografovaný snímok vyhovuje, tak bude prenesený na obrazovku detailu rany. Na tejto obrazovke by sa mali nachádzať základné informácie o aktuálnom snímku rany (ako napríklad meno rany, vypočítaná veľkosť, dátum vykonania snímania a poznámka) a samotná snímka znázorňujúca ranu s jej ohraničením. Na túto obrazovku by sa mal užívateľ dostať aj keď si v detaile liečby vyberie danú konkrétnu fotku rany. Návrh je možné vidieť na obrázku 3.6.

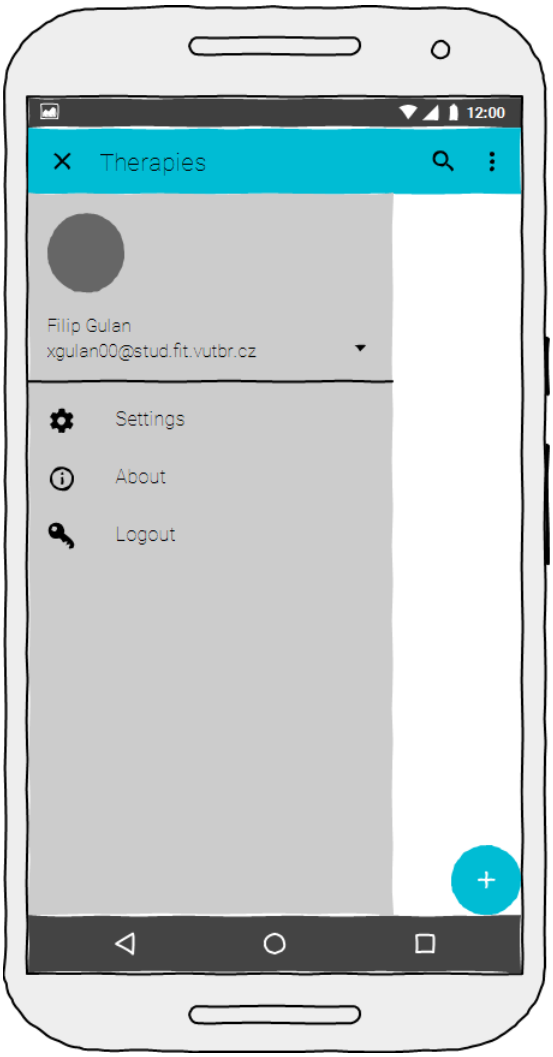
Obrázok 3.7 zobrazuje návrh globálneho menu, ktoré by malo byť dostupné z každej obrazovky. Toto menu by sa malo dať otvoriť z vrchného panelu aplikácie. Malo by obsahovať základné informácie o aktuálne prihlásenom užívateľovi, užívateľ by sa tu mal mať možnosť odhlásiť a mala by tu byť možnosť sa z tohoto menu dostať na obrazovku nastavení.



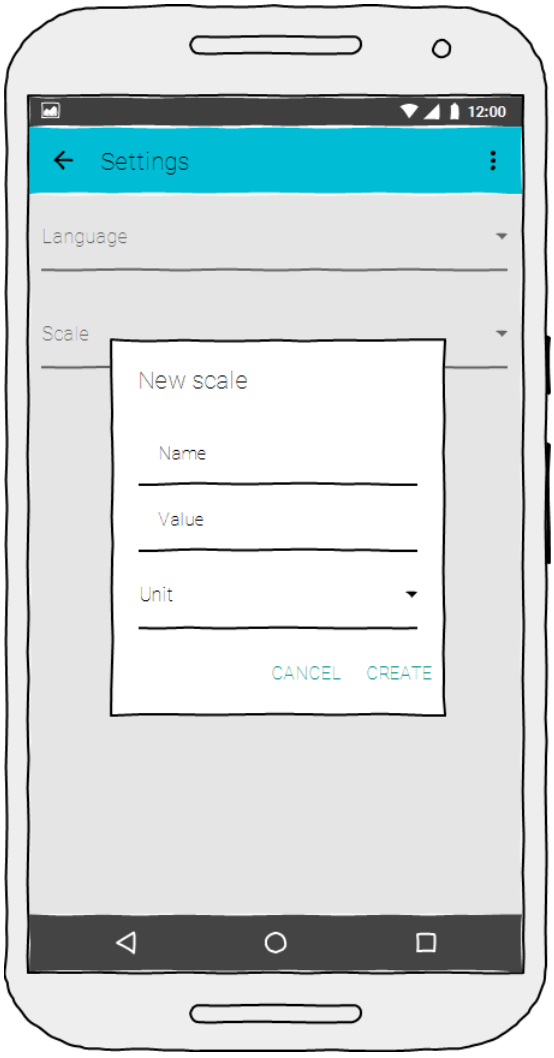
Obr. 3.5: Obrazovka náhľadu práve získanej a detekovanej rany

Obr. 3.6: Obrazovka detailu vyfotenej rany

Na obrazovke nastavení zobrazenej na obrázku 3.8 by sa mali dať spravovať predovšetkým jednotlivé mierky. Mali by sa tu dať pridávať, editovať a vymazávať pomocou kontextového okna. Zároveň by mala byť prítomná možnosť výberu mierky, ktorá sa aplikuje ako východzia.



Obr. 3.7: Globálne menu



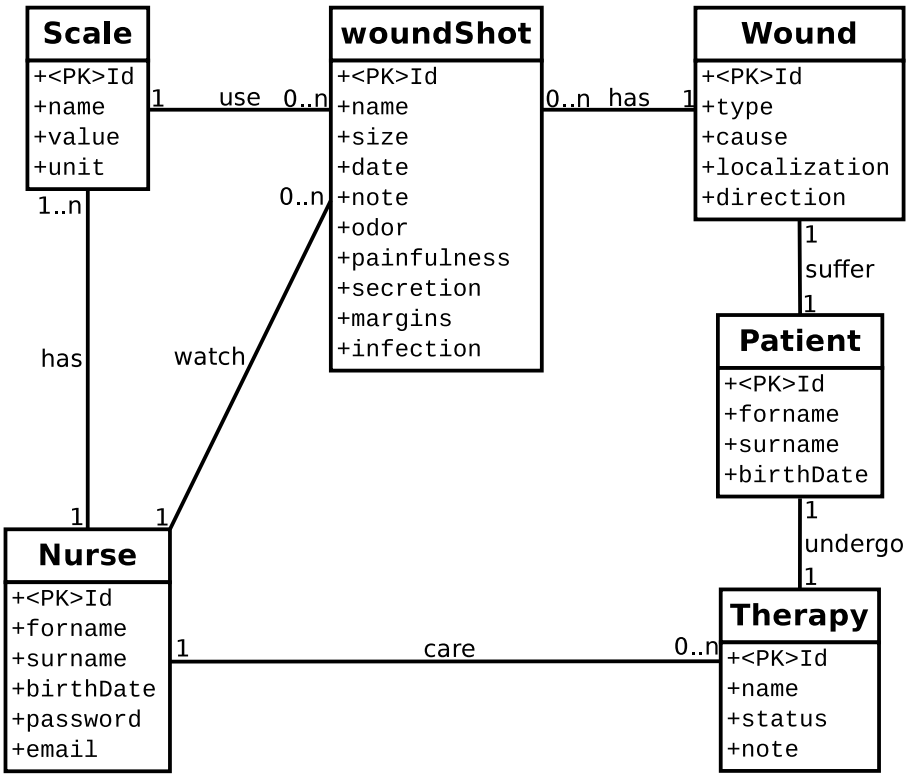
Obr. 3.8: Obrazovka nastavení a pridávanie mierky

3.3 Návrh databáze

Návrh databáze bol spracovaný pomocou Entity Relationship diagramu, ďalej iba ER diagram, s Unified modeling notáciou. Tento diagram sa používa v softwarovon inžinierstve pre konceptuálne a abstraktné znázornenie dát. Pri modelovaní ER diagramu vzniká jeden z typov konceptuálnych, alebo schematických dátových modelov systému a požiadavok naň z hora dole. V prípade, že sa jedná o návrh informačného systému založeného na databáze a aplikácia tejto diplomovej práce môže byť chápaná ako jednoduchý informačný systém, tak sa tento konceptuálny model v neskoršej fázy mapuje na logický dátový model a ten ďalej na fyzický dátový model.

Návrh využíva informácie z predchádzajúcej kapitoly o Návrhu aplikácie a samotné návrhy spolu úzko súvisia, keďže funkcionalita pracuje s dátami, ktoré sa získavajú z databáze a zároveň sa do nej ukladajú. Informácie sú tu však podané mierne odlišne, tak aby z nich bolo možné namodelovať ER diagram a uvedomiť si vzťahy medzi jednotlivými entitami. Jednou z hlavných entít figurujúca v tomto databázovom návrhu je liečenie. Liečenie ob-

sahuje svoj názov, aktuálny stav priebehu (ako napríklad dokončené, prebiehajúce, alebo iné). V každom liečení je vždy zapojený jeden konkrétny pacient. A každé liečenie prebieha vždy pod dohľadom jedného konkrétneho ošetrovateľa. Entita pacient obsahuje svoje meno, priezvisko a dátum narodenia. Okrem toho, že pacient podlieha jednému liečeniu, ako už bolo spomenuté, tak každý pacient má jednu ranu, ktorá ho trápi. Táto rana v sebe obsahuje svoj typ, príčinu rany, lokalizáciu a smer rany. Tieto 3 entitné množiny (terapia, pacient a rana) sú modelované ako slabé, z toho dôvodu, že vo výsledku budú brané ako jeden databázový záznam a v diagrame sú oddelené iba pre prehľadnosť návrhu. Nepredpokladá sa, že bude existovať X pacientov s X ranami. Skôr sa predpokladá jeden pacient, na ktorom bude sledovaná jedna rana. A ak by sa náhodou našiel pacient s viac ranami, tak sa iba pridá ďalšie liečenie, čo síce bude znamenať miernu redundanciu dát, ale vo výsledku bude aplikácia jednoduchšia na užívateľské rozhranie, keďže sa nebude musieť vždy vyberať zo zoznamu pacientov, poprípade pridávať nových do tohoto zoznamu. Každá rana môže obsahovať niekoľko svojich snímok a každá snímka patrí práve jednej rane. Každá táto snímka obsahuje svoj názov, veľkosť, dátum snímania, poznámku v prípade potreby. Ďalej snímka obsahuje či je aktuálne rana bolestivá, či zapácha, či sa na nej nachádza sekrét, či obsahuje infekciu a aké sú jej okraje. Každý tento snímok používa vždy jednu konkrétnu mierku a je sledovaná jedným konkrétnym ošetrovateľom. Ošetrovateľ, ktorý môže sledovať niekoľko týchto snímok rán v sebe obsahuje svoje meno, priezvisko, dátum narodenia, svoje heslo k systému a emailovú adresu. Ošetrovateľ môže vlastniť niekoľko mierok, ale minimálne vždy jednu. Poslednou entitou diagramu je mierka. Mierka obsahuje svoje meno, hodnotu a jednotku a vždy patrí práve jednému ošetrovateľovi, ale využívať ju môže niekoľko snímok. Výsledný ER diagram je možné vidieť na obrázku 3.9.



Obr. 3.9: ER diagram návrhu databáze

3.4 Návrh aplikačného rozhrania

Aplikačné rozhranie by malo slúžiť ako prostredník medzi samotnou aplikáciou a databázou, ale hlavne tiež by malo slúžiť pre spracovávanie snímok chronických rán, ich detekciu, lokalizáciu a určenie plochy. Celé aplikačné rozhranie nachádzajúce sa na servery by malo byť implementované v duchu REST architektúry. Aplikácia by sa mala dotazovať na server a získavať z neho odpovede. Rozhranie by malo obsahovať 4 základné HTTP metódy ako je GET, POST, PUT a DELETE. Metóda GET by slúžila pre prístup k dátam na uložených na servery, napríklad pre získanie zoznamu liečení, alebo konkrétneho liečenia určeného svojim identifikátorom. Metóda POST by mala slúžiť pre vkladanie dát na server, napríklad pre vytváranie nového liečenia. Metóda PUT by mala slúžiť pre modifikáciu dát na servery, napríklad pre editáciu informácií o liečení. A posledná metóda DELETE by mala slúžiť pre vymazávanie dát zo serveru, napríklad odstránenie liečenia. Príklady všetkých metód na konkrétnom príklade liečení spolu s navrhnutými url adresami, na ktoré by sa malo dať dotazovať je možné vidieť v tabuľke 3.1, pričom so všetkými ostatnými dátami by sa malo pracovať rovnako (napríklad pacienti, snímky...).

Akcia	Metóda	Obsah Body	URL
Získať zoznam liečení	GET	NULL	/therapy
Získať liečenie s ID 34	GET	NULL	/therapy/34
Vytvoriť nové liečenie	POST	{name: "Name", status: Äctive"}	/therapy
Editovať liečenie s ID 34	PUT	{name: ”New Name"}	/therapy/34
Odstrániť liečenie s ID 34	DELETE	NULL	/therapy/34

Tabuľka 3.1: Príklady pre manipuláciu s liečením prostredníctvom aplikačného rozhrania.

Každý dotaz by mal byť úplne definovaný a mal by obsahovať všetky potrebné informácie k jeho úspešnému vykonaniu. V prípade neúspechu by malo rozhranie vrátiť tomu odpovedajúci špecifický chybový kód. Hlavné a najbežnejšie rozšírené HTTP status kódy je možné vidieť v tabuľke 3.2.

Kód	Význam
200 OK	úspešná operácia a v odpovedi sa nachádzajú dáta
201 Created	operácia vkladania dát skončila úspechom
204 No Content	úspešná operácia, ale žiadne dáta sa nenachádzajú v odpovedi
400 Bad Request	server nerozumie požiadavku, kvôli nesprávnej syntaxi
401 Unauthorized	server pre prístup k dátam vyžaduje autorizáciu
403 Forbidden	server odmietol vykonať požiadavku
404 Not Found	na servery sa nenašli požadované dáta
500 Internal Server Error	server zaznamenal neočakávanú udalosť

Tabuľka 3.2: Najbežnejšie používané status kódy HTTP v REST.

Kapitola 4

Použité technológie

Ako už názov tejto kapitoly napovedá, kapitola predstavuje a popisuje technológie, ktoré boli vybrané vrámci návrhu a následne použité pri samotnej programovej implementácii práce. Celá implementácia, ako aj samotná kapitola, je rozdelená do logických celkov. Podľa týchto celkov bolo potrebné vybrať vhodný programovací jazyk, prípadne aplikačný rámec, pre serverovú časť (aplikačné rozhranie), klientskú časť (aplikácia) a zároveň samotnú databázu.

4.1 Klientská časť - aplikácia

Klientskou časťou je v prípade tejto práce označená samotná hybridná mobilná aplikácia, poprípade aplikácia spustiteľná na desktopovom zariadení. Hybridný prístup k tvorbe aplikácie bol zvolený namiesto klasického natívneho prístupu z dôvodu, že takto vytvorené aplikácie sú vysoko multiplatformné v zmysle, že ich stačí naprogramovať raz a už iba s minimálnym zásahom do kódu pobežia takmer na väčšine bežných operačných systémov. Bol zvolený hybridný aplikačný rámec Ionic verzia 2, ktorý je postavený na programovacom jazyku Typescript (nadmnožina jazyka Javascript) a na aplikačnom rámci Angular2. Typescript sa v rámci Ionic používa na logiku aplikácie. Pre vizuál aplikácie sa používa klasické HTML a CSS, alebo preprocesor SCSS. Táto práca je zameraná na platformu Android a platformu Windows. Platformu Android je možné pokryť pomocou technológie Apache Cordova, Windows zase pomocou technológie Electron.

- **HTML -** Hyper Text Markup Language je momentálne najpoužívanejší značkovací jazyk pre tvorbu webových stránok. HTML pre popis dokumentu používa značkovacie tagy, ktoré hovoria danému programu, internetovému prehliadaču, ako má daný dokument štruktúrovať. Originálne bol tento jazyk vyvinutý na popis štruktúry dokumentov, hlavne nadpisov, paragrafov, alebo zoznamov, ktoré sa zdieľali medzi inštitúciami a vedeckými pracovníkmi. Postupom času sa avšak z HTML stal celosvetovo rozšírený štandard pre popis štruktúry webových stránok. V minulosti sa HTML používalo nielen pre popis štruktúry, ale aj pre popis jeho výzoru. V súčastnosti sa výzor webových stránok definuje pomocou CSS. [13]

- **CSS -** Cascading Style Sheet je jednoduchý dizajnérsky jazyk, ktorý popisuje ako budú dané HTML tagy zobrazené na obrazovke počítača, papieri, alebo inom médiu. Medzi hlavné výhody CSS patrí to, že šetrí čas, pretože štýl napísaný v CSS je možné potom ďalej používať na iné webové stránky a zároveň sa zrýchľuje načítanie stránok,

keďže sa štýl nedefinuje väčšinou po jednom tagu, alebo po celej triede tagov. V priebehu rokov sa z CSS, tak ako to bolo aj s HTML, stal štandard pre webové stránky a podporujú ho všetky moderné webové prehliadače. [22]

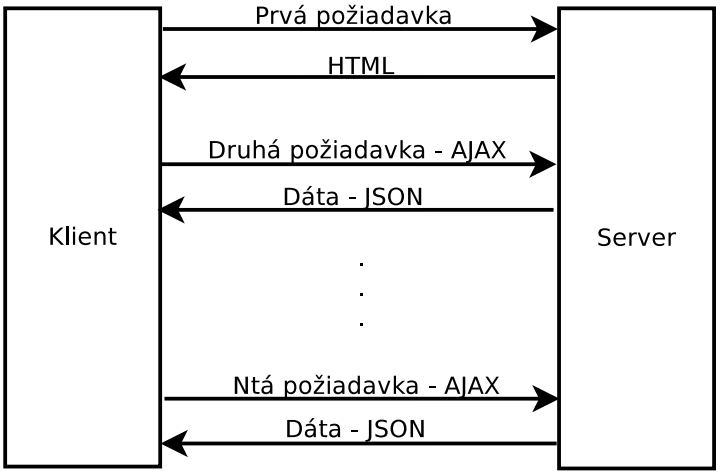
- **SCSS -** Syntactically Awesome Stylesheet je CSS preprocesor, ktorý rozširuje bežné CSS o celú radu vylepšení. Za zmienku stoja napríklad premenné, zanorovanie pravidiel, alebo importy. To má za následok, že kód je oveľa viac organizovaný, prehľadnejší, rozsahovo menší a je možné dosiahnuť netriviálnych cieľov oveľa jednoduchšie. Jedná sa v podstate o nadmonžinu CSS. Varianta syntakticky viac podobná klasickému CSS sa nazýva SCSS (Sassy CSS). [19]

- **Javascript -** je interpretovaný objektovo-orientovaný programovací jazyk, ktorý je známy predovšetkým ako skriptovací jazyk používaný na webových stránkach. V kontexte webových stránok sa používa predovšetkým pre dynamickú manipuláciu, ako napríklad odozvy na rôzne udalosti, animácie a iné. Je spúšťaný webovým prehliadačom. V súčastnosti sa rozširuje aj mimo klientskú stranu webu a je ho možné vidieť ako serverové riešenie v podobe stále viac populárneho Node.js. Je založený na vytváraní prototypov. Podporuje imperatívne, ale aj funkcionálne programovanie. [1]

- **Typescript -** Ako aplikácia napísaná v Javascripte rastie, tak začína byť neprehľadná a ťažko udržiavateľná. Z toho dôvodu Microsoft vyvinul Typescript. Typescript je vlastne nadmnožina Javascriptu, ktorá nezahŕňa iba samotný jazyk, ale aj početnú sadu nástrojov. Všetok kód napísaný v Typescripte sa transpiluje do čistého Javascriptu. Typescript rozširuje obyčajný Javascript o silné statické typovanie a je plne objektovo orientovaný (je možné použiť dedičnosť, triedy, rozhrania a iné veci typické pre čisto objektovo orientované jazyky). V neposlednom rade je v ňom možné používať všetky bežné Javascriptové knižnice. [21]

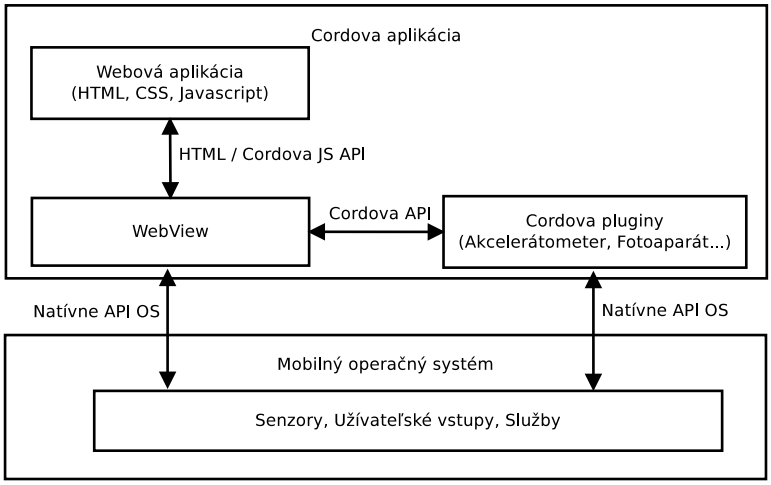
- **Angular2 -** je Typescriptový aplikačný rámec pre tvorbu webových tzv. Single Page aplikácií. Je vyvinutý týmom z Googlu, ktorý pracoval aj na originálnom AngularJS. Aj keď Angular2 a AngularJS majú podobné princípy, prakticky ide o dva odlišné aplikačné rámce. Tento rámec bol prvýkrát predstavený v roku 2014. Jeho myšlienka je založená na tvorbe znovupoužiteľných komponentov a v podstate využíva o Model Controler View architektúru, ktorú je ale možné vymeniť napríklad za architektúru Redux, známu hlavne z Facebook rámca React. [2]

  Single page aplikácie sú webové aplikácie, ktoré používajú väčšinou server iba ako zdroj a úložisko dát. Získané dáta sú následne vykresľované na klientovi pomocou Javascriptu. Pri prvom príchode na takúto stránku sa najprv stiahnu potrebné Javascriptové súbory a základná kostra HTML. Následne sa vykonajú operácie, ktoré určia, čo sa má zobraziť. Podľa toho čo sa má zobraziť je asynchrónnym dotazom (AJAX) na server požiadané o dáta, ktoré prídu v odpovedi typicky vo formáte JSON. Pri ďalšej interakcií so stránkou sa robia vždy len dotazy na samotný obsah. Server je v prípade takýchto webových stránok pasovaný iba do jednoduchého rozhrania, ktoré spracováva požiadavky z klienta a prípadne získava a ukladá dáta. [34]



Obr. 4.1: Klient-server komunikácia v Single page aplikácii

- **Cordova -** je rámec, ktorý zabalí HTML, CSS a Javascript aplikáciu do natívneho kontaineru, WebView, ktorý vo výsledku vyzerá a chová sa ako bežná natívna aplikácia. Kontainer dokáže pristupovať k funkciám zariadenia vďaka natívnym zásuvným modulom a Javascriptovému aplikačnému rozhraniu. Cordova podporuje celú škálu platforiem od Android, cez iOS, až po Windows Phone. Projekt bol najprv vytvorený firmou Nitobi, ktorá bola v roku 2011 kúpená známou firmou Adobe Systems. Adobe Systems následne uvoľnila kód Cordovy. [3]



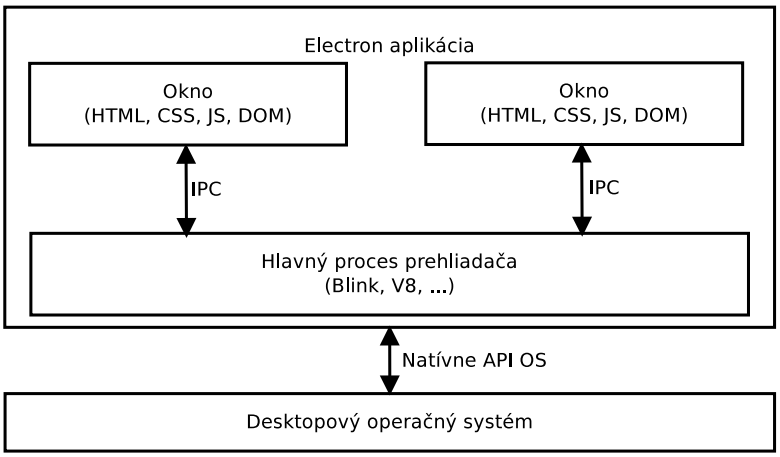
Obr. 4.2: Architektúra aplikácie vytvorenej pomocou Cordova technológie

- **Ionic2 -** je kompletné SDK pre vývoj hybridných mobilných aplikácií. Celý aplikačný rámec je postavený na Angular2 a rozširuje ho o ďalšie komponenty a funkcionalitu typickú pre mobilné zariadenia. Pomocou Cordova zásuvným modulom a ich implementácií v tomto rámci je možné tiež pristupovať k natívnym funkciám zariadenia. [14]

  Hybridné mobilné aplikácie sú aplikácie, ktoré na prvý pohľad vypadajú ako natívne aplikácie, avšak ich programovanie prebiehalo väčšinou pomocou webových technológií

s následným zabalením pomocou technológie Cordova alebo inej podobnej. Medzi najväčšiu výhodu takéhoto prístupu patrí to, že aplikáciu stačí napísať iba raz a potom ju je možné už iba s drobnými zmenami vydávať na rôzne mobilné platformy.[23]

- **Electron -** je knižnica vyvinutá GitHubom pre tvorbu multiplatformných desktopových aplikácií pomocou webových technológií, ktoré už boli popísané. Electron je ponúkaný pod licenciou MIT a má otvorený kód. Využíva kombináciu technológie Chromium a Node.js, ktoré zabalí spolu s webovou aplikáciou do jednej výslednej aplikácie. Takúto výslednú aplikáciu je možné spúšťať na platforme Windows, Mac a tiež Linux. K natívnym funkciám operačného systému je možné pristupovať, tak ako v prípade Cordovy, pomocou natívnych aplikačných rozhraní. [9]



Obr. 4.3: Architektúra aplikácie vytvorenej pomocou Electron rámca

4.2 Databázová časť

Pre uchovanie dát na strane serveru bola zvolená možnosť uloženia do databáze. Táto možnosť bola zvolená kvôli tomu, že manipulácia s dátami je pomerne rýchla, keďže sa do pamäte neukladá celá databáza, ale iba jej požadovaná časť. Okrem toho, pre získanie dát sa dá použiť pokročilejší dotazovací jazyk a je podporovaný prístup viacerých užívateľov súčasne.

Ako databáza bol zvolený typický zástupca NoSQL databáz menom MongoDB, ktorý je vyvíjaný MongoDB industries a má otvorený zdrojový kód. Dáta ukladá v štruktúre podobnej JSON dokumentom, a to v štruktúre Binary JSON, ktorý samotný JSON obohacuje o ďalšie typy. Súvisiace informácie sú uložené spolu pre rýchly dotazovací prístup pomocou MongoDB dotazovacieho jazyka. Používa dynamické schémy, to znamená, že záznamy nemajú pevne danú štruktúru a v priebehu sa táto štruktúra môže meniť. Pomocou tohoto dátového modelu je možné reprezentovať hierarchické vzťahy a ukladať polia a iné komplexnejšie dátové štruktúry. MongoDB dbá na to, aby bola zabezpečená vysoká dostupnosť a škálovateľnosť. Nespornou výhodou NoSQL a konkrétne MongoDB oproti Mysql je, že vývoj aplikácie s MongoDB je jednoduchší, keďže sa dokumenty v databáze prirodzene mapujú do moderných objektovo orientovaných jazykov a odpadá tak objektovo relačné mapovanie objektov. Obrovskou výhodou oproti napríklad MySQL je už spomínaná škálovateľnosť.

Samotné MongoDB vyžaduje pre svoje fungovanie iba adresár na uloženie dát a teda môže byť ľahko distribuované spolu s programom. [25]

```
[
    item1ID: {
      name: 'MongoDB',
      description: 'NoSQL'
    },
    item2ID: {
      name: 'MySQL',
      description: 'Relational'
    }
]
```

Zdrojový kód 4.1: Ukážka štruktúry NoSQL MongoDB databáze.

4.3 Serverová časť - aplikačné rozhranie

Serverová časť pokrýva všetky operácie, ktoré sa nevykonávajú priamo v zariadení, ale vykonávajú sa vzdialene na strane serveru. Medzi takéto operácie patrí napríklad ukladanie dát do databáze, autentifikácia, autorizácia užívateľa alebo samotná práca so snímkom chronickej rany. Serverová časť v tejto práci tvorí vzdialené aplikačné rozhranie. Je tvorená pomocou skriptovacieho jazyka Python, za pomoci aplikačného rámca Flask. Flask vo svojom jadre dodržiava princípy REST a je vďaka tomu možné jednoducho navrhnúť plne funkčné RESTful aplikačné rozhranie, s ktorým v tejto práci priamo komunikuje klientská aplikácia. Pre spracovanie obrazu, konkrétne samotných fotiek chronických rán, bola vybraná Python implementácia populárnej knižnice na spracovanie obrazu zvaná OpenCV.

- **REST -** Representational state transfer je súbor niekoľkých prísnych, ale jasne definovaných pravidiel pre návrh architektúry distribuovaného systému, hlavne rozhrania medzi jednotlivými komponentami, poprípade serverovou a klientskou časťou. Základným stavebným kameňom RESTu je zdroj. Zdroj môže byť čokoľvek, napríklad konkrétny objekt v databáze, dokument a podobne. Základným pravidlom je, že každý objekt musí mať svoju URL. Klient nikdy nevidí zdroj priamo, ale vždy iba jeden z jeho obrazov, ktorý je nazývaný reprezentácia. Reprezentácia je strojovo čitateľná reprezentácia aktuálneho stavu zdroja. Môže to byť obrázok, HTML stránka a podobne. Výber reprezentácie môže byť ovplyvnený klientom pomocou riadiacej informácie, alebo serverom na základe klasifikácie klienta. Samotná komunikácia cez REST je bezstavová, čo znamená, že každý požiadavok obsahuje všetky informácie potrebné k jeho vykonaniu. REST definuje 4 základné operácie: POST pre vytváranie, GET pre získavanie, PUT pre zmenu a DELETE pre vymazávanie. [24, 32]
- **Python -** je open source objektovo-orientovaný vysokoúrovňový skriptovaci jazyk, ktorý vytvoril Guido van Rossum v rokoch 1985-1990. Kód Pythonu spadá pod licenciu GNU General Public License. Python je interpretovaný a teda nie je nutné programy v ňom napísané prekladať. Programy sú namiesto prekladu priamo vykonávané interpreterom. Okrem toho je taktiež interaktívny, to znamená, že užívateľ môže priamo interagovať s interpreterom a písať programy aj týmto spôsobom. Ďalšou nespornou výhodou Pythonu je to, že je rozšíriteľný a nízko-úrovňové moduly sa dajú písať v programovacom jazyku C. Vďaka tomu môže byť dosiahnuté lepšej efektivity. [18]

Python bol zvolený najmä kvôli tomu, že je pomerne rozšírený, bežne sa používa na serverovú časť webových informačných systémov, má na výber z veľkého množstva intuitívnych aplikačných rámcov, dobre si rozumie s relačnými, ale aj s NoQSL databázami a v neposlednom rade je preň dostupná implementácia OpenCV.

- **Flask -** je aplikačný mini rámec, ktorý je napísaný v programovacom jazyku Python. Slúži pre vývoj webových aplikácií alebo vo všeobecnosti serverových častí informačných systémov, poprípade aplikačných rozhraní. V základe má k dispozícii iba funkcie s prácou pre URL, HTTP a šablónami. Z toho plynie, že je jednoduchý na osvojenie a pre účely tejto práce úplne postačuje. [10]

- **OpenCV -** je knižnica počítačového videnia. Je napísaná v jazyku C a C++ a je možné ju použiť na všetkých bežných aktuálnych platformách. Rozhranie je dostupné pre Python, Java, Ruby a ostatné bežné programovacie jazyky. OpenCV bolo navrhnuté pre výpočtovú efektivitu so silným dôrazom na aplikácie bežiace v reálnom čase. Optimalizácie knižnice boli vykonávané na všetkých úrovniach, od algoritmov až po viac jadrové inštrukcie procesorov. Hlavným cieľom je ponúknuť jednoduchú infraštruktúru počítačového videnia, ktorá by pomáhala ľuďom tvoriť zložitejšie aplikácie oveľa rýchlejšie a jednoduchšie. Samotná knižnica obsahuje viac ako 500 funkcií, pokrývajúce rôzne odvetvia vedy. OpenCV v sebe obsahuje okrem svojej samotnej implementácie aj pod-knižnice pre strojové učenie. keďže počítačové videnie a strojové učenie spolu úzko súvisia. Pod-knižnice sa z veľkej časti orientujú na rozpoznávanie vzorov a rozdeľovanie do zhlukov. [26]

Kapitola 5

Implementácia

Táto kapitola pojednáva o samotnej programovej realizácií a približuje implementačné detaily tejto diplomovej práce. Kapitola vychádza z informácií uvedených v predchádzajúcich kapitolách, presnejšie popisuje prevod návrhov vykonaných v kapitole 3 na konkrétne programové riešenie, pri ktorom boli použité technológie popísané v kapitole 4. Rovnako, ako aj kapitola o analýze a návrhu riešenia, tak aj táto kapitola je rozdelená do niekoľkých logických celkov. Najprv je popísaná implementácia klientskej, či už mobilnej, alebo desktopovej aplikácie. V poradí druhou časťou je popis finálnej formy databáze, ktorá je ihneď nasledovaná popisom implementácie serverového aplikačného rozhrania, pod ktoré spadá aj práca so snímkami. Celá kapitola je na záver ukončená popisom vybraných významnejších implementančných detailov ako je napríklad prihlasovanie, získavanie snímkov, alebo generovanie PDF súborov.

5.1 Implementácia klientskej aplikácie

Aplikácia, ako sa píše už v kapitole 4, je implementovaná pomocou rámca pre vývoj hybridných mobilných aplikácií menom Ionic. Funkčnosť na mobilnom zariadení so systémom Android zabezpečuje rámec Cordova a na desktopovom zariadení so systémom Windows zase rámec Electron.

5.1.1 Štruktúra aplikácie a popis jej komponentov

Celá aplikácia je umiestnená v zložke *src* a pozostáva z niekoľkých typov tried. Každý typ triedy je umiestnený vo vlastnom priečinku, ktorý je pomenovaný podľa tohoto typu. Konkrétne sa jedná o priečinok *app*, *effects*, *enum*, *model*, *pages*, *providers* a priečinok *reducers*. Okrem týchto typov priečinok *src* obsahuje ešte konfiguračné súbory spolu so zložkou *assets* a *theme*. Priečinok *assets* obsahuje všetky statické súbory, ktoré sa netranspilujú [1], konkrétne sa jedná o ikonu aplikácie spolu s obrázkom tlačidla na prihlasovanie Google účtom. Priečinok *theme* obsahuje súbory, poprípade iba jeden súbor, ktorý predstavuje globálnu tému aplikácie, ktorá v sebe drží definície farieb, definície štýlov písma a iné podobné nastavenia globálneho charakteru.

V zložke *app* sa nachádza hlavná komponenta a modul, ktorý predstavujú celú aplikáciu. V *app* module sa nachádzajú všetky definície používaných stránok, poskytovateľov služieb, reduktorov, efektov, modálnych okien a poprípade, ak by sa v tejto práci využívali, aj

[1] Transpilácia je proces prekladu zdrojového kódu daného jazyka na zdrojový kód iného jazyka, v prípade práce z Typescriptu na Javascript [39]

direktív a rúr. Každá komponenta, a nie je tomu inak ani pri *app* komponente, sa skladá vždy z troch hlavných typov súborov. Z typescript súboru, z html súboru a z scss súboru.

- Typescript súbor predstavuje akýsi kontrolér danej komponenty poprípade jej logiku.
- Html súbor predstavuje pohľad a obsahuje zobrazované dáta jednosmerne, alebo obojsmerne naviazané na dáta v typescript súbore.
- Scss súbor definuje vzhľad zobrazovaného html dokumentu, avšak aplikácia v tejto práce si vystačí s bežnými komponentami Ionic rámca a tento súbor je preto často prázdny.

App komponenta je špeciálna tým, že je globálna a existuje po celú dobu aplikácie a preto obsahuje hlavné menu a udalosti a funkcie viazané hlavne k tomuto hlavnému menu. Vďaka tomuto menu je možné prechádzať medzi stránkami aplikácie, alebo sa odhlasovať.

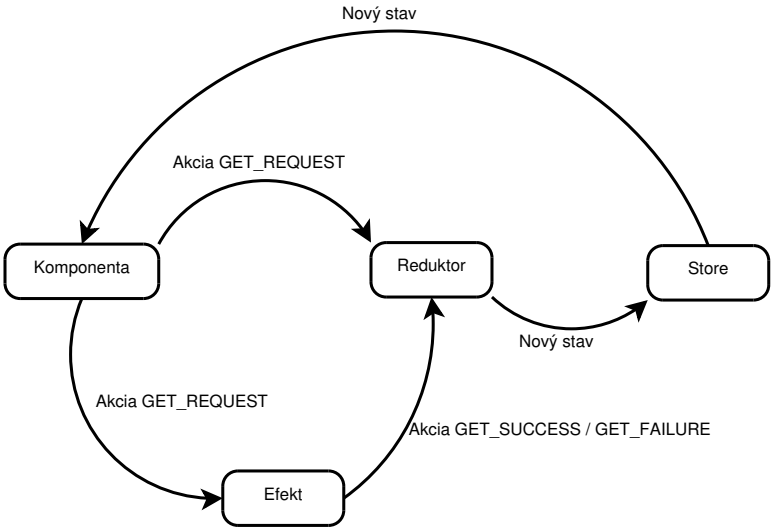
V priečinku *models* sa nachádzajú modely, alebo konkrétne rozhrania, všetkých hlavných entít použitých v tejto práci, menovite to sú *scale.model*, *therapy.model*, *user.model* a *wound.model*. Tieto rozhrania zabezpečujú typovú kontrolu daných entít a okrem toho behom vývoja aplikácie zabezpečovali inteligentné našeptávanie. Mimo samotných modelov, sa v tomto priečinku nachádza ešte súbor *app.state*, ktorý predstavuje hlavný stav aplikácie a taktiež môže byť chápaný, ako akýsi model. *App.state* slúži pre prácu s *ngrx storom*, jeho efektmi a reduktormi. Celý stav aplikácie je vďaka nemu rozdelený na niekoľko podstavov určitého typu podľa vlastností tohoto *app.state modelu*. Ide napríklad o typ *therapy*, *wound*, alebo iný a je vďaka tomu umožnené pracovať iba s vybranou časťou celkového stavu aplikácie.

V priečinku *enums* sa zase nachádzajú všetky vymenúvacie typy, poprípade konštanty, ktoré sa v rámci aplikácie nemenia a sú tu definované nemenne, napevno. Ide napríklad o typy mierok, alebo stavy liečenia.

Priečinok *reducers* obsahuje reduktory, ktoré slúžia na manipuláciu so stavom aplikácie. Stav aplikácie je vlastne nemenná dátová štruktúra. Stav aplikácie sa teda nemodifikuje, ale nahrádza sa vždy novým stavom. Na zmenu stavu, alebo presnejšie povedané jeho nahradenie, slúžia akcie, ktoré presne popisujú a definujú, ako sa daný stav zmení. Tieto zložky dohromady tvoria *ngrx store*. Ngrx store je vlastne kontajner, alebo menežment stavu aplikácie založený na RxJS určený pre Angular a je inšpirovaný stále viac populárnym Redux. Hlavnou myšlienkou je umožnenie vytvárania komponentov schopných používať stratégiu detekcie zmeny nazvanú *OnPush*. Informácie boli prebrané z [17]. Komponenty teda načúvajú zmene stavu aplikácie, alebo jeho časti a následne na to reagujú. Všetky súbory reduktorov v sebe obsahujú okrem funkcie na zmenu stavu aj definíciu spomínaných akcií. Stav tejto aplikácie sa skladá, ako už bolo niekoľko krát spomenuté, z niekoľkých častí. Konkrétne sa jedná o časť zoznamu mierok, zoznamu liečení, zoznamu rán a o časť jednotlivej terapie a jednotlivej rany. Veľkou výhodou takéhoto prístupu je, že stačí odoslať konkrétnu akciu na *ngrx store* a zmenené dáta automaticky dostanú všetky komponenty, ktoré čakajú na zmenu danej časti stavu. Tento prístup sa ukázal ako veľmi vyhovujúci, nakoľko aj pri nie až tak rozsiahlej aplikácií, ako je aplikácia tejto práce, vznikalo početné množstvo závislostí, kedy zmena dát na nejakom mieste vyžadovala ešte aj zmenu dát na niekoľkých iných miestach.

Efekty, ktoré sú umiestnené v rovnomennej zložke *effects* sú taktiež súčasťou *ngrx store*. Jedná sa vlastne o akési vedľajšie účinky odoslania akcie, kedy sa nemení stav aplikácie, ale pred zmenou stavu aplikácie sa odošle ešte napríklad požiadavka na aplikačný server a získajú sa dáta. Dokončenie tejto akcie vyvolá zmenu stavu, veľmi často zmenu stavu

pomocou už získaných dát zo serveru. Informácie boli čerpané z [16]. Obrázok 5.1 popisuje konkrétny prípad chovania v aplikácií, kedy sa vyvolá akcia *GET_REQUEST*. Túto akciu zachytí ako reduktor, tak aj efekt. Reduktor na akciu zareaguje tak, že iba vráti rovnakú kópiu stavu. Avšak efekt na akciu zareaguje tak, že sa vykoná požiadavka na aplikačný server a podľa úspechu/neúspechu operácie sa vyvolá akcia *GET_SUCCESS*/*GET_REQUEST* s dátami a tento nový stav je potom uložený do *ngrx store*. *Ngrx store* nakoniec tento stav vráti nejakej načúvajúcej komponente/komponentám.



Obr. 5.1: Tok dát aplikácie využívajúcej ngrx store

Poskytovatelia nachádzajúci sa v priečinku *providers slúžia*, ako názov napovedá na poskytovanie dát, či už nemenne uvedených v triede, alebo získaných z aplikačného serveru. Okrem získavania dát môžu obsahovať funkcie používané v rôznych častiach aplikácie a tak odprosťovať od písania opakovaných úsekov kódu. Jedná sa spravidla o triedy návrhového vzoru singleton[2], teda vzoru, ktorý sa v rámci aplikácie nachádza iba raz a inštancie týchto tried sú držané v *dependency injection*[3] kontajnere aby k nim mohli pristupovať komponenty a tak využívať ich funkcionalitu a dáta. V tejto aplikácií sa využíva celkom 5 poskytovateľov.

- Api poskytovateľ (*api.provider*) slúži na komunikáciu medzi aplikáciou a aplikačným serverom. Obsahuje metódy na vytváranie, mazanie a získavanie zdrojov serveru podľa daných parametrov.
- Auth poskytovateľ (*auth.provider*) slúži na autentifikáciu a operácie k tomu príbuzné. Konkrétne obsahuje metódy na prihlasovanie, zaregistrovanie, odhlasovanie, zistenie stavu prihlásenia a taktiež drží informácie o aktuálne prihlásenom užívateľovi.
- Poskytovateľ dočasných náhradných dát (*mock-data.provider*) slúžil počas vývoja na poskytovanie predpripravených dát pre rýchlejší vývoj, kedy sa pracovalo s týmito konštantnými dátami namiesto reálnych dát.

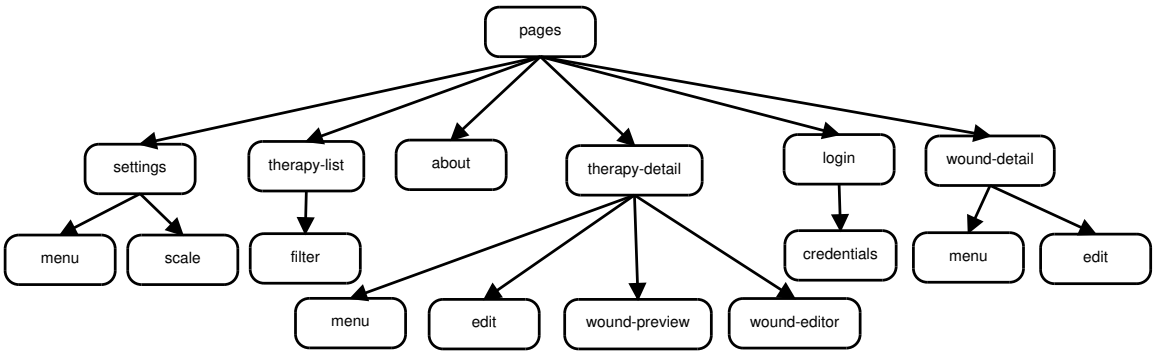
[2]Singleton je návrhový vzor, ktorého podstatou je vytvorenie iba jednej inštancie triedy, ktorá sa používa naprieč celou aplikáciou.[37]

[3]Dependency injection je návrhový vzor, v ktorom trieda dostáva svoje závislosti z externých zdrojov namiesto toho, aby si ich vytvárala sama.[7]

- Poskytovateľ elementov užívateľského prostredia (*ui-elements.provider*) zapúzdruje niektoré často používané funkcie samotného Ionic rámca. Ide napríklad o vytváranie a zobrazovanie grafických užívateľských elementov nazvaných *Toast*, alebo *Alert*. Funkcie tohoto poskytovateľa takto obsahujú funkcie rámca a presnejšie špecifikujú chovanie týchto funkcií pomocou parametrov.

- Poskytovateľ všeobecných funkcií je posledným poskytovateľom. Tento poskytovateľ obsahuje funkcie, ktoré neboli zaradené do žiadneho iného poskytovateľa. Konkrétne tento poskytovateľ obsahuje iba 2 funkcie, a to funkciu na odstránenie vlastnosti *id* a mapovanie *_id->$oid* na *id*. Funkcie sú prítomné preto, že na strane serveru a na klientskej strane sa pracuje s identifikátormi odlišne. Je to paradoxne pre jednoduchšiu prácu na klientovi, pretože databáza pracuje so svojou špecifickou štruktúrou identifikátorov a tá na klientskej strane nebola potrebná.

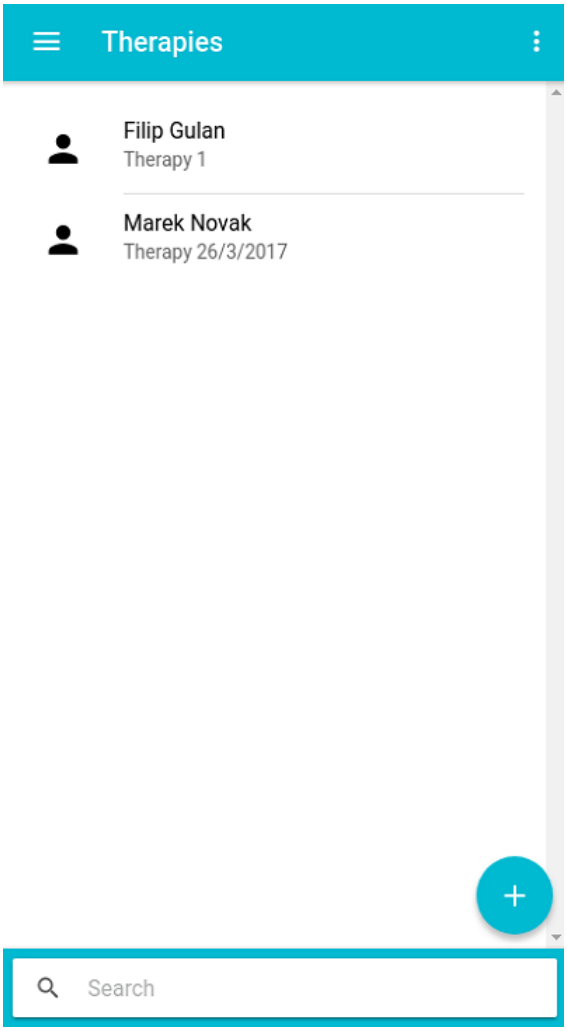
Posledným a zároveň aj najdôležitejším, najbohatším priečinkom na informácie a funkcionalitu je priečinok *pages*. Tento priečinok obsahuje všetky stránky, obrazovky aplikácie. V koreni adresára sa nachádzajú vždy priečinky hlavných stránok. V týchto priečinkoch sa nachádzajú potom okrem samotných zdrojových súborov stránok ešte aj ďalšie stránky, alebo komponenty, ktoré súvisia s danou nadradenou stránkou sa sú v nej v podstate vnorené. Takýmto príkladom môže byť modálne okno, ktoré síce vyzerá ako stránka, ale je to vo svojej podstate súčasť inej stránky. Každá stránka, ako už bolo spomenuté pri *app* module, sa skladá z niekoľkých súborov. Z typescript súboru plniaceho funkciu kontroléru. Z scss súboru, ktorý definuje výzor danej stránky a je v aplikácií v podstate skoro nepoužívaný, keďže sa používajú zväčša neupravené Ionic komponenty už v úhľadnej Material design forme. Z html súboru, ktorý predstavuje pohľad na dáta z kontroléru a definuje štruktúru stránky. A nakoniec z modulu, ktorý je prítomný iba aby bolo možné prepojiť stránky s hlavným *app* modulom. Samotná aplikácia obsahuje 6 hlavných stránok. Ide o stránku obsahujúcu informácie o aplikácií (*About*), stránku slúžiacu na prihlásenie (*Login*), stránku obsahujúcu nastavenie a vytváranie mierok (*Settings*), stránku obsahujúcu zoznam liečení (*Therapy list*), stránku zobrazujúcu detail liečenia (*Therapy detail*) a nakoniec stránku obsahujúcu detail rany, samotnú konkrétne zachytenú snímku (*Wounbd detail*). Štruktúru členenia stránok je možné vidieť na obrázku 5.2.



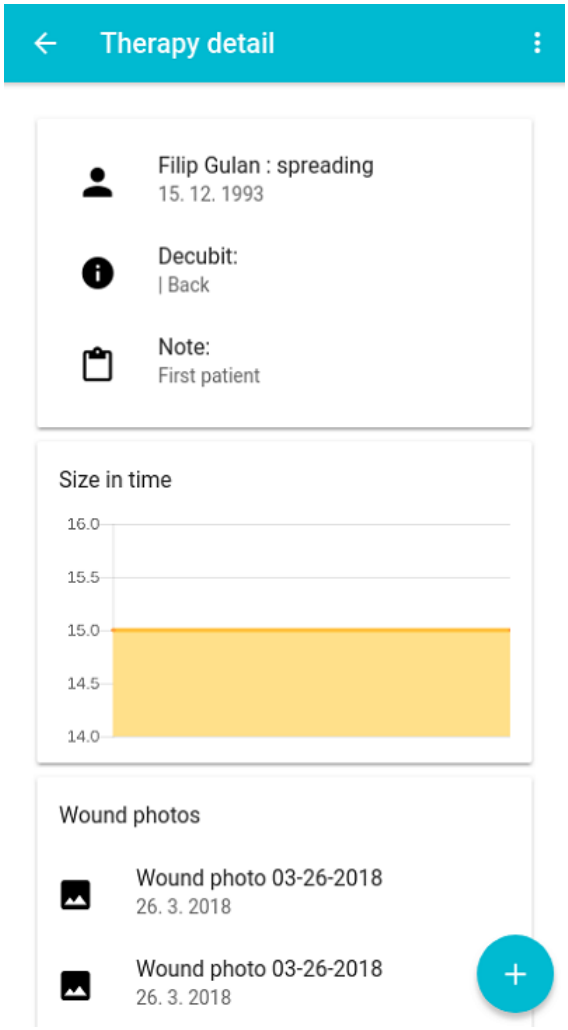
Obr. 5.2: Štruktúra členenia stránok a ich komponentov

Z obrázka je teda jasne vidieť, aké komponenty a stránky obsahujú hlavné stránky. Je zreteľne vidieť, že väčšina stránok obsahuje komponentu *menu*, poprípade *filter*, ako to je pri stránke zoznamu liečení, čo je vlastne kontextové menu umiestnené v pravom hornom rohu hlavičky aplikácie. Okrem komponenty *menu* väčšina stránok tiež obsahuje podstránku

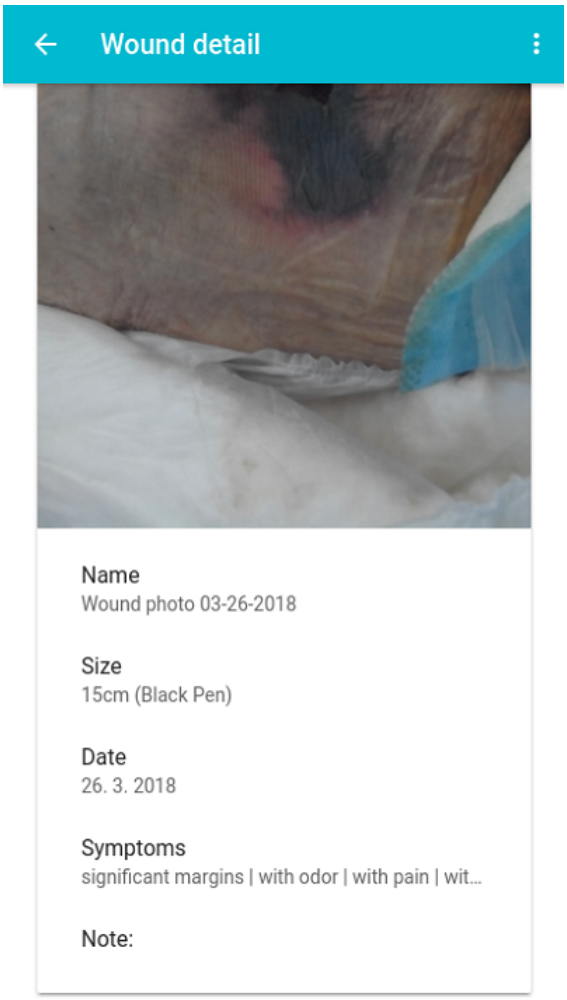
edit. Táto podstránka je vlastne modálne okno slúžiace na vytváranie, poprípade editovanie daného vybraného zdroja. Implementované stránky je možné vidieť na obrázkoch 5.3 až 5.6.



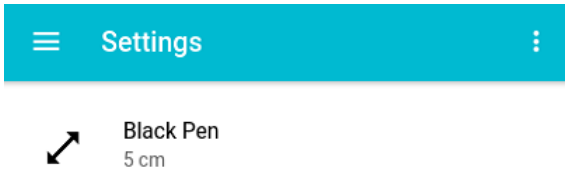
Obr. 5.3: Obrazovka zoznamu liečení



Obr. 5.4: Obrazovka detailu liečenia



Obr. 5.5: Obrazovka detailu rany



Obr. 5.6: Obrazovka nastavení, zoznamu mierok

5.1.2 Implementačné detaily aplikácie

Klientská aplikácia obsahuje niekoľko prvkov, ktoré stoja za zmienku a ktoré sú z hľadiska realizácie riešenia zaujímavé. Ide o nasledujúce implementačné detaily:

- **Filtrovanie -** Filtrovanie sa nachádza v zozname liečení. Zoznam štádií liečenia, podľa ktorých je možné filtrovať je implementovaný ako samostatná komponenta *Therapy Filter* typu tzv. *PopOver*, ktorá je dostupná ako kontextové menu v pravom hornom rohu stránky. Toto menu je tvorené v rodičovskej komponente, teda v zozname liečení. V tejto rodičovskej komponente sa tiež načúva na udalosť zavretia tejto komponenty, ktorá pri svojom uzavretí vracia dané vybrané štádium. Zavretie komponenty nastáva pri výbere možnosti z filtra užívateľom. Po získaní daného štádia z komponenty sa následne nad originálnym poľom dát použije metóda *filter()*, ktorá porovnáva štádia jednotlivých prvkov s užívateľom vybraným štádiom a tak užívateľ na výstup dostáva iba prvky, ktoré spĺňujú podmienky filtru.

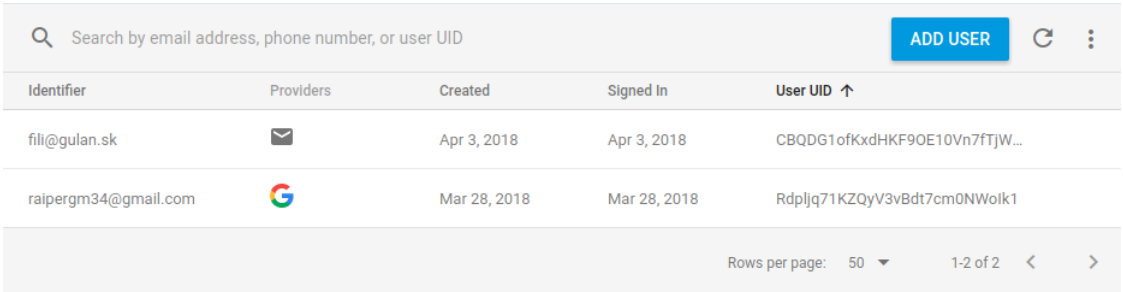
- **Vyhľadávanie -** Vyhľadávanie sa znovu nachádza na stránke zoznamu liečení, tak ako filtrovanie, a je umiestnené dole v pätičke stránky. V podstate sa jedná iba o jednoduché textové pole, ktoré je nastavené na odchytávanie zmeny svojej hodnoty a pri každej takej udalosti sa vykoná filtrovanie pomocou metódy *filter()* nad originálnym poľom dát. Konkrétne sa odfiltrovávajú položky, ktoré vo svojom názve, v krstnom mene pacienta a priezvisku pacienta neobsahujú podreťazec z vyhľadávacieho pola.

- **Graf -** Jediné grafy, ktoré sa v aplikácií nachádzajú sú grafy zobrazované v rámci stránky detailu liečenia. Na vykresľovanie grafov bola zvolená knižnica *Chart.js*, ktorá je veľmi jednoduchá na použitie, dostatočne minimalistická a pritom obsahuje všetko čo sa od knižnice takéhoto typu očakáva. Dokáže vytvoriť 6 najbežnejších typov grafov, ktoré vykresľuje na plátno (značka *canvas*) a všetky takéto grafy sú plne responzívne, čo účel práce vyžaduje, keďže aplikácia je určená pre mobilné a aj desktopové zariadenia. Po tom, ako sú dáta získané z aplikačného servera, sa volá metóda *createGraph()*, ktorá vykoná vyžadované nastavenia grafu a naplní graf získanými dátami. Keďže sa nejedná o knižnicu originálne vyvinutú pre Angular, alebo Ionic a pracuje s natívnym elementom, tak musel byť element plátna v html sprístupnený pre Ionic komponentu pomocou dekorátoru *ViewChild*.

5.2 Finálna forma databáze

Dátový model databáze predstavený v rámci jej návrhu bol v priebehu programovej realizácie upravený a prispôsobený tak, aby mohol byť uložený v NoSQL databáze akou je MongoDB.

Prihlasovanie do aplikácie sa deje pomocou Firebase Authentification, keďže požiadavky aplikácie vyžadovali prihlásenie pomocou Google účtu a súčasne táto metóda prihlasovania je preferovaná a rozšírená v aplikáciách bežiacich na operačnom systéme Android. Z tohoto dôvodu sú dáta uložené nielen vo vlastnenej databáze, ale niektoré typy dát boli distribuované do Firebase Authentification úložiska. Táto vzdialená databáza drží práve dáta patriace k zdravotným sestrám, ošetrovateľom, doktorom, alebo jednoducho vo všeobecnosti všetkým užívateľom aplikácie.



Obr. 5.7: Ukážka záznamu vo Firebase Authentification

Na obrázku 5.7 je možné vidieť takýto záznam jedného užívateľa vo Firebase Authentification úložisku. Takýto záznam vo Firebase Authentification úložisku pozostáva z Indentifikátora, ktorý je v prípade tejto práce vždy e-mail, z poskytovateľa, ktorý môže byť momentálne buď *Google*, alebo jednoducho *E-mail/Password* (v budúcnosti pripadá do úvahy *Facebook*, *Twitter* a iné), dátumu vytvorenia, dátumu posledného prihlásenia do

aplikácie a užívateľovho unikátneho identifikačného čísla slúžiaceho pre jednoznačnú identifikáciu v rámci aplikácie.

Liečenie, snímka rany, mierka sú potom typy dát, ktoré sú držané práve vo vlastnenom dátovom úložisku. Okrem týchto spomenutých kolekcií sa nachádza v databáze kolekcia *session*, ktorá predstavuje aktuálne sedenie užívateľa. To znamená, že táto kolekcia si drží základné informácie o aktuálne prihlásenom a v aplikácií pracujúcom užívateľovi. Dané sedenie si v sebe vždy drží token, získaný prihlásením pomocou Firebase, unikátny identifikátor užívateľa získaný znova z Firebase a ešte platformu, na ktorej je sedenie platné (buďto *core* pre internetový prehliadač, *core-electron* pre Electron desktop program, alebo *cordova-mobile-android* pre mobilnú Android aplikáciu). Výsledné kolekcie je možné vidieť v kóde.

```
wound_database {
    scale: [
        {
            _id: 'scale-id-1',
            value: '1',
            unit: 'px',
            name: 'Scale 1',
            user: 'firebase-user-id-1',
        }
        ...
    ],
    session: [
        {
            _id: 'session-id-1',
            userId: 'firebase-user-id-1',
            platform: 'core',
        },
        ...
    ],
    therapy: [
        {
            _id: 'therapy-id-1',
            userId: 'firebase-user-id-1',
            name: 'Therapy 1',
            type: 'Decubit',
        },
        ...
    ],
    wound: [
        {
            _id: 'wound-id-1',
            therapy: 'therapy-id-1',
            user: 'firebase-user-id-1',
            originalLocation: '/home/filip/images/therapy-id-1.jpg'
            date: '154789',
            ...
        },
        ...
    ],
}
```

Zdrojový kód 5.1: Finálna štruktúra databáze

Kód obsahuje vždy iba pár nevyhnutných položiek, ktoré stačia na predstavu, ako je to v databáze uložené. O to, že sa v kolekcií nachádzajú ďalšie dokumenty daného typu informujú tri bodky. V tomto kóde je možné si všimnúť, ako nakoniec boli realizované vzťahy

medzi jednotlivými entitami. Napríklad vzťah medzi *therapy* a *wound* je realizovaný pomocou odkazov, kedy *wound* obsahuje v sebe položku *therapy*, ktorá odkazuje na dané konkrétne liečenie podľa unikátneho identifikátoru tohoto liečenia. Vzťahy na základe referencií namiesto zanorovania štruktúr, ktoré NoSQL podporuje, boli zvolené kvôli tomu, že v podstate všetky ďalšie vzťahy sú iba vzťahy medzi entitou vo vlastnej databáze a používateľom vo Firebase Authentification úložisku a chcelo sa zachovať akýsi jednotný štandard práce s dátovými entitami. MongoDB, ako aj iné databázy, umožňuje uložiť obrázky priamo do databáze. Pre intuitívnejšiu prácu s aplikačným serverovým rozhraním sa do databáze ale ukladajú iba cesty k súborom, ktoré sú uložené v priečinkoch na serveri.

5.3 Implementácia serverového aplikačného rozhrania

Aplikačné rozhranie REST nachádzajúce sa na strane serveru bolo postavené na rámci Flask a ďalších pomocných knižníc programovacieho jazyka Python. Celá programová realizácia tohoto rozhrania bola rozdelená do niekoľkých samostatných súborov a každý tento súbor predstavuje vlastne danú triedu zabezpečujúcá konkrétnu sadu operácií. Realizácia obsahuje dokopy 6 súborov, z toho 1 súbor plniaci funkciu hlavnej funkcie, ktorá je mienená ako hlavný prístupový bod a 5 samostatných tried umiestnených v balíčku *api_utils*.

- Trieda *Auth* v súbore *auth.py* zabezpečuje iba jednu konkrétnu funkcionalitu a to overenie tokenu, teda či daný prichádzajúci token, ktorým sa preukazuje daný prihlásený užívateľ patrí naozaj k nejakému účtu vo Firebase Authentification.

- Súbor *database.py* obsahujúci triedu *Database* zabezpečuje zase všetku prácu s MongoDB databázou za pomoci funkcií a metód knižnice pymongo. V tejto triede sa vytvára spojenie s databázou identifikovanou jej menom na danom servery a porte. V rámci tejto práce sa jedná o server s adresou *127.0.0.1* teda *localhost*, keďže aplikačné rozhranie a databáza sa nachádzajú na rovnakom stroji, port *27017* a databázu *wound_database*. Okrem toho obsahuje metódy na prácu s dokumentami a kolekciámi tejto databáze.

- Súbor *httpStatusCode* obsahuje Hypertext Transfer Protokol stavové kódy serveru, ktoré sa využívajú pre zlepšenie čitateľnosti kódu. Sú v tejto triede uložené ako konštanty.

- Trieda *Pdf* uložená v súbore *pdf.py* slúži na generovanie Portable Document Format súborov s dátami získaných z databáze. V triede sa nachádzajú 2 šablóny v HTML formáte podľa ktorých sa generuje výsledný pdf súbor. Jedna šablóna je pre dáta o liečení a druhá pre dáta o rane. V pdf súbore liečenia sa nachádza okrem textových informácií ešte aj graf priebehu liečby, ktorý je najprv generovaný pomocou knižnice matplotlib a uložený ako obrázok. Tento obrázok sa následne vkladá do pdf suboru počas jeho tvorby. Tvorbu pdf súborov zabezpečuje knižnica pdfkit. Celý podrobný proces tvorby pdf súborov na strane serveru a prepojenie s klientskou stranou sa nachádza ďalej v kapitole.

- Poslednou triedou, a to jednoznačne v rámci tohoto diplomového projektu aj najvýznamnejšou triedou, je trieda *Vision*. Trieda je uložená znovu v rovnomennom súbore *vision.py*. Zabezpečuje všetko okolo spracovania snímky rany, detekcie rany zobrazenej na snímke, detekcie mierky a vypočítania plochy rany na základe mierky a rany samotnej. Všetky tieto detekcie a výpočty sú realizované pomocou knižnice OpenCV.

Spracovaniu snímky, detekcií a výpočtu plochy je ešte v tomto texte venovaný značný ďalší priestor, keďže ide o významnú časť práce.

Prístupovým bodom do aplikačného rozhrania, ktorý spracováva požiadavky klientov a následne im odpovedá je súbor *main.py*. V tomto súbore sú definované všetky prístupové koncové adresy k zdrojom serveru a všetky možné metódy týchto adries. Pri požiadavke na adresu serveru bez akýchkoľvek ďalších ciest k zdroju pomocou metódy *GET* je vrátený reťazec *"Wound detector API V1"*, ktorý slúži na overenie, či na danej doméne beží naimplementované aplikačné rozhranie. Pri požiadavke na adresu */static/priečinok/meno-súboru* je vrátený nahraný súbor podľa svojho *mena* nachádzajúci sa na serveri v danom *priečinku*. Adresa */auth/platforma* slúži na prihlasovanie a odhlasovanie užívateľa podľa zvolenej metódy na danej *platforme*. Metódou *POST* sa užívateľ do aplikácie prihlasuje, metódou *DELETE* naopak odhlasuje. Ostatné adresy vo formáte */zdroj* slúžia na prístup k určenému zdroju a k manipulácií s týmto zdrojom na základe zvolenej metódy. *GET* pre prístup k zdroju. *POST* pre vytvorenie nového zdroju. Adresy vo formáte */zdroj/id* slúžia na prístup ku konkrétnemu zdroju identifikovanému pomocou unikátneho identifikátoru. Metódou *GET* sa takýto zdroj získa, metódou *PUT* modifikuje a metódou *DELETE* odstráni.

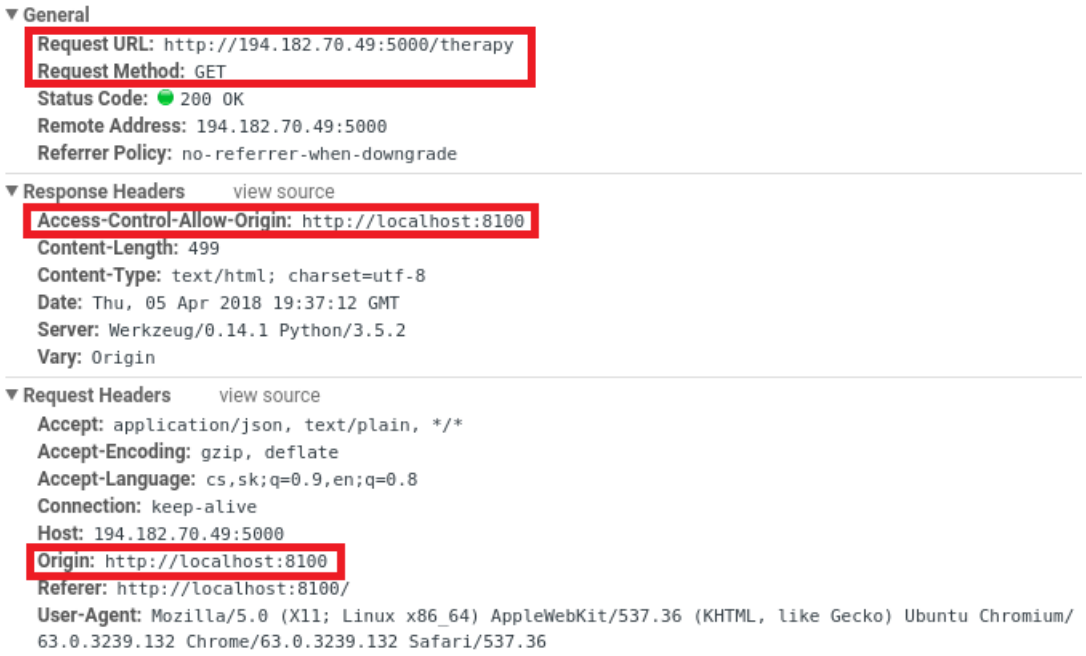
5.3.1 Cross-Origin Ressource Sharing

Odosielanie požiadaviek na server je limitované tzv. *Same origin policy* a teda nie je možné prijímať odpovede na požiadavky z inej domény, ako je doména, z ktorej bolo požiadané. Túto veľmi známu limitáciu rieši *Cross-origin resource sharing*, ďalej iba *CORS*. CORS je vlastne mechanizmus, ktorý povoľuje zakázané zdroje na internetovej stránke, a tak aplikácie alebo internetové stránky môžu prijímať aj odpovede na požiadavky z inej domény, ako je doména, na ktorej sa aplikácia, alebo internetová stránka nachádza. Pre použitie musí nastať iba jednoduchá modifikácia hlavičiek v odpovedi serveru. Prehliadač musí posielať hlavičku *Origin* nastavenú na aktuálnu doménu aplikácie, poprípade webovej stránky. Server musí posielať hlavičku *Access-Control-Allow-Origin*, s hodnotou konkrétnej povolenej domény (domény aplikácie, alebo internetovej stránky), ktorá môže prijímať odpovede zo servera. Pre povolenie všetkých domén bez rozdielu sa používa *, čo je pre serverové rozhranie tejto práce chcené, keďže aplikácia bude bežať v rôznych zariadeniach a na rôznych platformách. Ďalej je možné ešte špecifikovať ďalšie nastavenia ako napríklad, ktoré *HTTP* metódy budú povolené. Toto je zabezpečené hlavičkou *Access-Control-Allow-Methods*, ale ani táto hlavička ani iné z rodiny *Access-Control-Allow-** nie sú v projekte využívané [33].

CORS teda funguje v nasledujúcich jednoduchých krokoch:

1. *CORS* požiadavku iniciuje klientská strana, aplikácia.
2. Prehliadač vloží do *HTTP* požiadavky hlavičky, spolu s hlavičkou *Origin*.
3. Server detekuje požiadavku, vytvorí odpoveď na túto požiadavku, vloží do nej *HTTP* hlavičky spolu s hlavičkou *Access-Control-Allow-Origin*, ktorá indikuje, že odpoveď je povolená na danej doméne a nakoniec túto odpoveď odošle prehliadaču.
4. Prehliadač detekuje, či je odpoveď na požiadavku povolená.
   (a) V prípade, že je požiadavka povolená prehliadač pošle dáta ďalej aplikácii alebo internetovej stránke.

(b) V prípade neexistencie hlavičiek v odpovedi, alebo neočakávania je odpoveď zamietnutá a klientská aplikácia si nemôže prečítať tieto dáta.



Obr. 5.8: Detail HTTP požiadavku a odpovede s hlavičkami *CORS*

Obrázok 5.8 zobrazuje náhľad na požiadavku vykonanú na server a odpoveď, ktorú server vrátil. Tento obrázok bol získaný z Chrome developer tools konzole, ktorá je dostupná v internetových prehliadačoch Chrome, alebo Chromium. Na obrázku je vidieť, že sa vykonávala požiadavka na adresu *http://194.182.70.49:5000/therapy* metódou *GET* za účelom získania zoznamu liečení. Ďalej je možné vidieť, že do požiadavky boli vložené rôzne *HTTP* hlavičky, medzi ktorými nechýba ani hlavička *Origin*. Po tom, ako server spracoval požiadavku, odoslal odpoveď so štandardnými hlavičkami. Okrem štandardných hlavičiek server do odpovede vložil hlavičku *Access-Control-Allow-Origin*. O povolenie mechanizmu *CORS* sa v rámci tejto práce stará knižnica navrhnutá špeciálne pre Flask rámec, menom Flask-Cors. Pre aktivovanie tejto knižnice stačí zavolať funkciu *CORS(app)* a knižnica už sama do každej odpovede pridáva požadované *CORS* hlavičky.

5.3.2 Poloautomatická detekcia chronickej rany

Spracovanie snímkov je komplexnejší proces, ktorý prebieha z časti na serveri, a z časti v klientskej aplikácií a je popísaný ďalej. V tejto sekcií kapitoly sa avšak pojednáva iba o samotnom algoritme pre poloautomatickú detekciu rany a o výpočte veľkosti detekovanej rany. Implementovaný algoritmus pre deteckiu rany bol inšiporvaný algoritmom uvedeným v odbornej publikácií [28]. Tento algoritmus požaduje pre svoj chod určenie bodu, ktorý sa nachádza vo vnútri rany. Tento proces začína volaním metódy *processImageAutomatic()*, do ktorej sa predajú súradnice počiatočného bodu. Behom algoritmu je vykonaných niekoľko v podstate samostatných krokov.

1. **Modifikovanie HSV -** HSV[4] model bol zvolený kvôli tomu, že môže byť modifikovaný do takej podoby, aby farby, ktoré sú pre ranu typické mali čo najväčšiu vzdialenosť. Pre zvýšenie presnosti pravdepodobnostnej mapy bol vytvorený modifikovaný HSV farebný model. Tento model napomáha reflektovať vzdialenosti medzi štyrmi farbami, ktoré sú v chronickej rane zahrnuté. Ide o červenú, žltú, čiernu a bielu farbu. V klasickom modeli je tmavo červená a tmavo žltá bližšie k čiernej a naopak bledo červená a bledo žltá bližšie k bielej, čo sa negatívne odzrkadľovalo na presnosti pravdepodobnostnej mapy. Jednotlivé kanále farebného modelu boli modifikované pomocou rovníc 5.1 až 5.3 [28]. Ako α bola experimentálne zvolená hodnota 9 a ako β zase -17, čo odpovedá 34 stupňovému uhlu. V kóde je tento prevod z originálneho HSV modelu na modifikovaný HSV model vykonaný pomocou funkcií *getHMod() getSVMod()*.

$$H_m = H + \beta \tag{5.1}$$

$$S_m = \frac{log(\alpha * S + 1)}{log(\alpha + 1)} \tag{5.2}$$

$$V_m = \frac{log(\alpha * V + 1)}{log(\alpha + 1)} \tag{5.3}$$

2. **Vytvorenie pravdepodobnostnej mapy RYKW -** Pravdepodobnostná mapa je v podstate matica slovníkov, ktoré obsahujú položky hodnôt pravdepodobnosti pre r (červená), y (žltá), k (čierna) a w (biela) a ďalšie položky pomocného charakteru. Položky *r, y, k, w* v podstate predstavujú s akou pravdepodobnosťou daný pixel prichlúcha k danej farbe. Mapa bola vytvorená tak, že každý HSV pixel $x_{i,j}$ bol prevedený na pixel modifikovaného farebného modelu HSV, a ten následne na jednotlivé pravdepodobnosti $RYKW$ podľa rovníc 5.4 kde $C_k = \{R, Y, K, W\}$ a $k = 1, 2, 3, 4$.

$$p_k(x) = \frac{1}{\left(\frac{d(C_k,x)}{d(R,x)}\right)^2 + \left(\frac{d(C_k,x)}{d(Y,x)}\right)^2 + \left(\frac{d(C_k,x)}{d(K,x)}\right)^2 + \left(\frac{d(C_k,x)}{d(W,x)}\right)^2} \tag{5.4}$$

$d(C_k, x)$ je vzdialenosť medzi pixelom x a nejakou špecifickou farbou C_k. Vzdialenosť medzi pixelom a danou farbou je daná rovnicami 5.5 až 5.8

$$d(R, x) = \sqrt{(H_m(x) - H_m(R))^2 + (S_m(x) - S_m(R))^2 + (V_m(x) - V_m(R))^2} \tag{5.5}$$

$$d(Y, x) = \sqrt{(H_m(x) - H_m(Y))^2 + (S_m(x) - S_m(Y))^2 + (V_m(x) - V_m(Y))^2} \tag{5.6}$$

$$d(K, x) = V_m(x) - V_m(K) \tag{5.7}$$

$$d(W, x) = \sqrt{(V_m(x) - V_m(W))^2 + (S_m(x) - S_m(W))^2} \tag{5.8}$$

kde $H_m(R) = 11/12$, $H_m(Y) = 1/12$, $V_m(R) = 1$, $V_m(Y) = 1$, $V_m(K) = 0$, $V_m(W) = 1$, $S_m(R) = 1$, $S_m(Y) = 1$, $S_m(W) = 0$ [28]. Matica pravdepodobnosti je v programe vytváraná pomocou funkcie *createProbabilityMap()* a jednotlivé pravdepodobnosti sú počítané pomocou funkcie *computeP()*.

[4]HSV je farebný model, ktorý pozostáva z 3 zložiek. Z farebného tónu, zo sýtosti farby a z hodnoty jasu. Model bol vytvorený v roku 1978 a autorom je Alvy Ray Smith.[12]

3. **Region Growing algoritmus -** Segmentácia rany je vykonaná pomocou algoritmu nazvaného Region Growing, ktorý funguje na princípe rozširovania okolia z nejakého počiatočného bodu a postupne do okolia zahrňuje susedné pixely [28]. Počiatočný bod je v tomto prípade zvolený manuálne užívateľom. Algoritmus prebieha v iteráciách a beží dokiaľ je možné spracovávať nejaké body. Končí teda vtedy, ak sú všetky body v susedstve spracované a žiadne ďalšie už neboli vygenerované, alebo v prípade, že sú všetky body obrázku spracované. Na začiatku je teda zvolený bod a identifikované jeho susedstvo. Pre každý bod susedstva sa zisťuje, či patrí do výslednej segmentácie alebo nie, a to tak, že sa vypočítava vzdialenosť medzi hodnotami pravdepodobnostnej mapy daného pixelu a priemernými hodnotami, ktoré sú vypočítavané z pixelov, ktoré už do segmentovaného okolia patria. Rovnicu vzdialenosti je možné vidieť v 5.9 kde p označuje bod z pravdepodobnostnej mapy a m priemernú hodnotu.

$$d(p,m) = \sqrt{(R_p - R_m)^2 + (Y_p - Y_m)^2 + (K_p - K_m)^2 + (W_p - W_m)^2} \tag{5.9}$$

V prípade, že zistená vzdialenosť je menšia ako daný prah (prah je experimentálne nastavený na hodnotu 0,035) tak je daný pixel zaradený medzi pixely segmentovanej rany a jeho susedstvo je pridané medzi body, ktoré sa budú znovu takto overovať. Týmto spôsobom sa získajú potenciálne všetky body, ktoré patria do okolia danej rany. Keďže algoritmus je pomerne časovo náročný, tak sa vykonáva na obrázku 4x zmenšenom oproti originálu. Takto sa dosiahlo toho, že vykonávanie algoritmu na testovacích snímkach trvalo od 5 sekúnd do 30 sekúnd. Tento čas bol priamo úmerný výslednej veľkosti segmentovanej rany, kedy sa na hranici 30 sekund segmentovala rana, ktorá zaberala odhadom tak 75 percent snímky. Vykonávanie tohoto algoritmu je v programe zaistené spustením funkcie *regionGrowing()*, generovanie susedstva pomocou funkcie *generatePoints()*, zistenie vzdialenosti bodu a priemeru pomocou funkcie *countDistance()* a nakoniec zistenie nového priemeru danej oblasti pomocou funkcie *setMeanProbability()*.

4. **Zistenie veľkosti rany -** Po tom, ako je rana segmentovaná pomocou algoritmu Region Growing a je tento priestor ohraničený určitou farbou (typicky bielou, alebo čiernou), tak je možné pomocou kontúr vypočítať plochu tejto rany. Obrázok sa teda najprv prevedie do stupňov šedi, následne sa prahuje a nakoniec sa použije funkcia knižnice OpenCV na hľadanie kontúr menom *findContours()*. Výsledná plocha je potom získaná tak, že sa iteruje nad kontúrami a najväčšej kontúre je zistená jej plocha pomocou OpenCV funkcie *contourArea()*. Takto sa teda získa plocha v pixeloch, ďalšie prepočty na veľkosť reálneho sveta sú vykonávané v klientskej aplikácií.

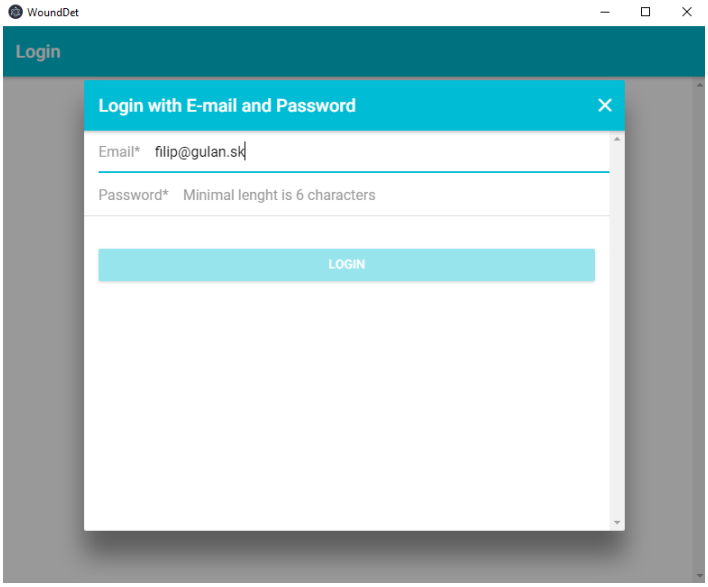
5.4 Implementačné detaily

Niektoré procesy sú vykonávané aj na strane klienta, to znamená v mobilnej, alebo desktopovej aplikácií, a aj na strane serveru v aplikačnom rozhraní. Medzi takéto procesy patrí napríklad prihlásenie užívateľa do aplikácie, získavanie snímkov, alebo generovanie PDF súborov. Všetky tieto procesy, ktoré sú vykonávané z časti na klientovi a z časti na serveri sú bližšie popísané práve v tejto kapitole.

5.4.1 Prihlasovanie

Prihlasovanie v aplikácií patrí ku komplexnejším procesom v rámci programovej realizácie. Tento proces je z časti vykonávaný vo Firebase Authentification, z časti na aplikačnom serverovom rozhraní a z časti v klientskej aplikácií. Je mu z toho dôvodu venovaná samostatná kapitola obsahujúca aj dovysvetľujúci obrázok 5.10.

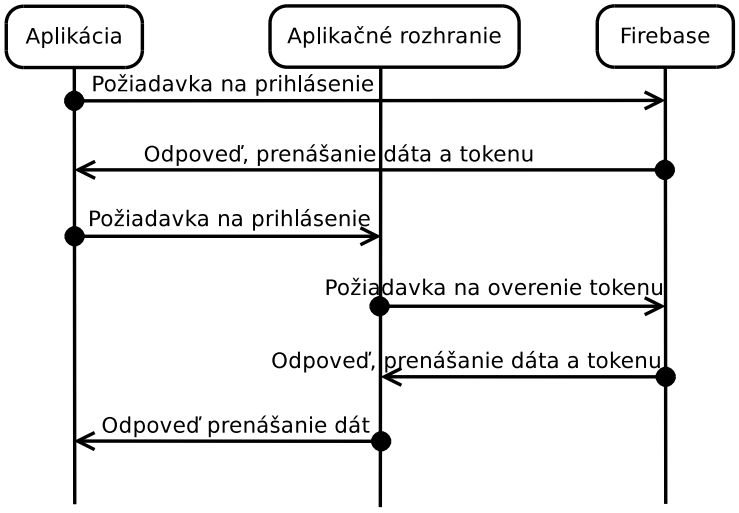
Firebase vo svojej momentálnej implementácií nepodporuje v Electron, v NW.js alebo v iných desktopových aplikáciach prihlasovanie pomocou Google účtu. Vo fórach a Github problémoch všetko naznačuje tomu, že v budúcnosti bude možné aj tento typ prihlasovania. Momentálne však hlavne kvôli bezpečnosti toto umožnené nie je. Tento problém bol vyriešený tak, že v desktopovej Electron aplikácií sa namiesto prihlasovania Google účtom prihlasuje iba pomocou e-mailu a hesla. Je avšak nutné upozorniť, že užívateľ zaregistrovaný a prihlásený e-mailom a heslom sa v žiadnom prípade nezhoduje s užívateľom vytvoreným a prihláseným pomocou Google účtu. Užívateľ zaregistrovaný Google účtom *novak@gmail.com*, nie je ten istý užívateľ ako *novak@gmail.com* zaregistrovaný e-mailom a heslom. Firebase Authentification týchto užívateľov vidí ako 2 rôznych, keďže sa jedná o 2 úplne odlišné typy prihlásenia, pričom užívateľ prihlasovaný e-mailom a heslom sa do aplikácie v podstate normálne registruje a jeho heslo neodpovedá heslu Google účtu ale heslu zvolenému pri registrácií. Na obrázku 5.9 je vidieť okno desktopovej verzie aplikácie zobrazujúce prihlasovanie pomocou e-mailu a hesla.



Obr. 5.9: Prihlasovacia obrazovka desktopovej verzie aplikácie

Prihlasovanie začína na prihlasovacej obrazovke/stránke. V prípade, že sa jedná o mobilnú Android aplikáciu, prípadne internetovú aplikáciu, tak je ponúknuté prihlasovanie aj pomocou Google účtu aj pomocou e-mailu a hesla. V prípade, že sa jedná o desktopovú verziu aplikácie, tak sa tam nachádza iba formulár slúžiaci na prihlasovanie pomocou e-mailu a hesla. Ide síce o 2 v podstate odlišné metódy prihlásenia, avšak vďaka Firebase Authentification ich prihlasovací cyklus v rámci aplikácie prebieha absolútne rovnako. Jediná odlišnosť je v tom, keď sa užívateľ prihlasuje prvý krát do aplikácie pomocou Google účtu, tak Firebase vytvorí vo svojej databázy záznam automaticky. Naopak pri metóde prihlasovania pomocou e-mailu a hesla sa tento záznam musí najprv vytvoriť programovo, teda pri prvom

prihlasovaní prebieha akási prvotná registrácia užívateľa, ktorú vykonáva samotná klientská aplikácia. Po stlačení tlačítka na prihlásenie Google účtom, alebo tlačítka na prihlásenia vo formulári, sa vykoná požiadavka na server Firebase Authentification, ktorá sa pokúsi užívateľa prihlásiť a v prípade úspechu vráti objekt užívateľa s ostatnými informáciami, ako je napríklad platný token. Získaný token sa následne odošle na serverové aplikačné rozhranie tejto práce na koncovú adresu */auth/platforma* metódou *POST*, kde platforma predstavuje platformu zariadenia, z ktorého sa užívateľ pokúša prihlásiť. Pred vykonaním požiadavky sa ešte nastaví *HTTP* hlavička *Authorization* na hodnotu získaného tokena. Na serveri je tento token overovaný, či sa jedná o platný token získaný z Firebase Authentification. Okrem samotného overenia je znovu získaný aj užívateľ, ktorému patrí tento token. V prípade, že je token úspešne overený, tak sa vytvorí v databázy v kolekcií *session* záznam predstavujúci dané sedenie identifikované svojim unikátnym identifikátorom a obsahujúce Firebase Authentification token, unikátny identifikátor prihláseného užívateľa a platformu z ktorej prihlásenie nastalo. Po tom, ako je token overený a je vytvorený dokument kolekcie *session*, tak je následne prihlásený užívateľ vrátený v odpovedi do aplikácie. Vrátený užívateľ sa následne uloží do poskytovateľa autentifikácie (*auth.provider*) a tieto informácie sú zobrazené v hlavnom menu, konkrétne je zobrazené meno, priezvisko, e-mail a fotka užívateľa. Po tom, ako bol užívateľ prihlásený v obidvoch častiach, aj vo Firebase Authentification časti a aj v časti serverového aplikiačného rozhania, je možné vykonávať dotazy na server pre získanie súkromných dát, medzi ktoré patria doslova všetky požiadavky serveru, ako napríklad získanie zoznamu liečení, zoznamu rán a podobne. Všetky požiadavky po procese prihlásenia teda obsahujú *HTTP* hlavičku *Authorization* s hodnotou tokenu z Firebase Authentification. Po požiadaní serveru o zdroj sa na serveri táto hlavička získa a porovnáva sa, či sa v databázy v kolekcií session nachádza takýto užívateľ. V prípade, že sa takýto užívateľ v kolekcií nenachádza, server vráti *401 UNAUTHORIZED*. V opačnom, v kladnom prípade je užívateľ získaný z databázy a jeho identifikátor je ďalej používaný pre získavanie žiadúcich zdrojov. Celý tento proces je pre názornosť zobrazený na obrázku 5.10.



Obr. 5.10: Priebeh prihlasovania

5.4.2 Generovanie PDF súborov

Ďalším procesom, ktorý presahuje rozdelenie na proces aplikácie a proces serveru je generovanie súborov typu Portable Document Format s informáciami o danom liečení, alebo o danej konkrétnej rane. Generovanie je možné zahájiť na stránke detailu liečenia v kontextovom menu, alebo v kontextovom menu na stránke detailu rany. Napriek tomu, že Javascript knižnice dokážu programovo vytvárať jednoduché PDF súbory, bola zvolená možnosť generovania týchto súborov na serveri. Rozhodnutie bolo vykonané hlavne kvôli obmedzeniam vo finálnej Android aplikácií, kde aj po početných pokusoch sa nepodarilo takto generovaný súbor správne uložiť do zariadenia. PDF súbory sa teda generujú na serveri. Proces generovania súboru iniciuje klientská aplikácia, alebo lepšie povedané užívateľ výberom možnosti generácie PDF v kontextovom menu. Po výbere tejto možnosti je úžívateľovi zobrazené okno načítania, ktoré je prítomné po celú dobu, až pokiaľ nie je súbor definitívne stiahnutý, alebo nenastala nejaká neočakávaná chyba. Následne je odoslaná na server požiadavka o generovanie súboru *HTTP* metódou *POST*. Server túto požiadavku príjme, získa informácie z databáze potrebné pre PDF súbor a pomocou knižnice PDFKit začne podľa HTML šablóny generovať samotný PDF súbor, ktorý uloží do priečinku *static/pdfs*. V prípade, že PDF súbor má obsahovať aj graf, konkrétne PDF súbory liečení obsahujú grafy, tak sa najprv tento graf vygeneruje a uloží sa ako Portable Network Graphic obrázok do priečinku *static/temp* odkiaľ sa potom použije v šablóne. Po skončení operácie na serveri je klientovi vrátený *HTTP* statu kód *200*. Týmto status kódom užívateľ vie, že PDF súbor je pripravený na stiahnutie. Sťahovanie samotného PDF súboru prebieha rozdielne pri Mobilnej a Desktopovej aplikácií. Pre stiahnutie súboru na platforme Android sa využíva natívny Cordova pluginu menom FileTransfer, ktorý súbor zo serveru stiahne a uloží do externého dátového priečinku, to je do priečinku *Android/data/net.raiper34.wounddetector/files*. Užívateľ je o tomto úspechu informovaný pomocou upozorňovacieho okna, v ktorom je zobrazená cesta k súboru. Aj v prípade desktopovej verzie nebolo možné stiahnuť súbor iba pomocou tradičných metód, ako napríklad pridanie atribútu *download* pre kotvový element. Namiesto toho sa musela vykonať požiadavka na server metódou *GET* a špecifikovať hlavička *responseType* nastavená na hodnotu *arrayBuffer*[5]. Dáta, ktoré boli následne získané zo serveru sa potom pomocou objektu *Blob*[6] a knižnice FileSaver uložili na užívateľom vybrané miesto. Keďže užívatel si sám volí, kde sa súbor uloží, tak o úspechu je informovaný obyčajnou *Toast* komponentou so základnou hláškou úspechu.

5.4.3 Získavanie snímkov

Medzi najvýznamnejšie procesy práce patrí jednoznačne aj proces vyhotovovania snímkov. Tento proces prebieha z časti v aplikácií a z časti na aplikačnom servery. Proces začína na stránke detailu konkrétneho liečenia, kde si užívateľ zvolí v rozkladacom plávajúcom tlačítku, či chce použiť už hotové snímky z lokálneho úložiska svojho zariadenia, alebo chce ranu rovno zosnímať pomocou mobilnej počítačovej kamery, poprípade fotoaparátu. Ak si užívateľ vyberie prvú možnosť a zvolí si už existujúcu snímku z lokálneho úložiska, tak sa táto snímka predáva modálnemu oknu ukážky rany pomocou *NavParams*. V druhom prípade sa iba prejde do rovnakého modálneho okna ale bez predania dát. V tomto modálnom okne sa v konštruktore detekuje, či boli predané nejaké dáta. V prípade existencie dát, sú

[5] ArrayBuffer objekt sa používa pre reprezentovanie všeobecnej vyrovnávacej pamäte surových binárnych dát s pevnou dĺžkou.[4]

[6] Blob objekt reprezentuje nemenné surové dáta, ktoré sú podobné súborovým dátam. Tieto dáta nemusia nutne reprezentovať natívny Javascript formát.[5]

tieto dáta zakódované do formátu *Base64*[7] a spôsobom, aký je bežný aj napríklad pre formulárové dáta tejto aplikácie, je obrázok odoslaný na aplikačný server. V prípade, že sa dáta do komponenty nepredali, tak sa v konštruktore aktivuje kamera a užívatel má tak možnosť zosnímať ranu priamo. Po vyhotovení sa snímka tejto rany znovu prekonvertuje do *Base64* formátu a odošle na aplikačný server. Aplikačný server príjme požiadavku obsahujúcu takto zakódovaný obrázok, obrázok prekóduje naspäť do originálneho kódovania, vytvorí dočasný dokument v kolekcií *wounds* a obrázok uloží do priečinku *static/shoots* s menom *identifikátoru* vytvoreného dočasného dokumentu v kolekcií *wounds*. Tento obrázok sa následne spracuje a prebehnú všetky požadované výpočty. Po úspechu tejto operácie je v aplikácií zobrazený daný spracovaný obrázok, tak že HTML element *img* obsahuje v atribúte *src* hodnotu adresy, ktorá sprostredkováva obrázky uložené na serveri (*/static/shoots/wound-id.jpg*). Zobrazením obrázku je užívateľ vyzvaný, či sa daný obrázok použije ako finálna snímka rany, alebo chce proces opakovať a použiť, alebo vytvoriť inú snímku. V prípade nevyhovujúcej snímky, kedy napríklad bola rana nesprávne detekovaná kvôli svetelným alebo iným podmienkam, tak užívateľ zvolí možnosť opakovať proces a znovu sa aktivuje kamera, s tým že dočasný obrázok je zo serveru vymazaný. V prípad, že obrázok vyhovuje, prejde sa do modálneho okna editácie/vytvárania snímky, kde užívateľ je nútený vyplniť konečné informácie potrebné pre vytvorenie snímky. Povinnou položkou v tomto formulári je iba meno, ktoré je prednastavené na *"Wound photo + aktuálny dátúm"* a položka mierky. Po vyplnení a odoslaní formulára je vytvorená konečne finálna snímka rany zobrazujúca sa v zozname snímok v detaile daného liečenia. Ak sa proces v ktoromkoľvek kroku preruší, tak je dočasný dokument z databázy spolu s dočasnou snímkou vymazaný.

5.4.4 Spracovanie snímkov

Detekcia a určenie veľkosti chronickej rany prebieha dvomi rôznymi metódami a je na užívateľovi, ktorú si vyberie. Rany je možné detekovať poloautomaticky, tak, že užívateľ označí jeden bod v rane, alebo je možné ranu "detekovať"ručne tak, že užívateľ ohraničí ranu čiarami a vytvorí tak uzavretý priestor, práve ktorého plocha sa bude následne zisťovať.

- **Poloautomatická detekcia -** Východzím módom, ktorý je pri detekcií zvolený je práve mód poloautomatickej detekcie. Užívateľ je v aplikácií nútený v podstate nastaviť iba počiatočný bod v priestore rany, pomocou ktorého sa bude rana detekovať. Tento počiatočný bod nachádzajúci sa na začiatku v strede, sa nastavuje ťahaním a pustením (tzv. drag and drop) zeleného kruhu nad zobrazovanou snímkou. Tento bod je vykresľovaný na plátne (element canvas), ktoré je priehľadné a umiestnené nad obrázkom. Manipulácia s týmto bodom je možná vďaka knižnici Fabric.js. Po umiestnení bodu je už možné spustiť vykonávanie samotnej poloautomatickej detekcie, kedy sa na server prenesú súradnice počiatočného bodu a spustí sa vykonávanie algoritmu, ktorý je popísaný pri implementácií aplikačného rozhrania.

- **Manuálna detekcia -** Manuálna detekcia je prítomná práve kvôli tomu, aby bolo možné zistiť veľkosť chronickej rany aj v prípade, kedy poloautomatická detekcia výrazne zlyhá a nie je možné v značne komplikovanej rane detekovať jej okolie. Manuálnu detekciu je nutné zapnúť v plávajúcom tlačítku v obrazovke editora rany. Po zapnutí tohoto módu je užívateľ schopný pomocou myši, alebo prsta (zálaží na danej

[7] *Base64* je kódovanie, alebo mechanizmus, ktorý povoľuje prenos binárnych dát cez média, ktoré umožňujú iba prenos textu. Vznikol kvôli potrebe pridávať do e-mailov obrázkový, video alebo iný binárny obsah. Kóduje vždy 3 oktety binárnych dát pomocou 4 *ASCII* znakov. [36]

platforme) ohraničiť ranu. Je to možné chápať ako nejaké kreslenie. Toto ohraničenie-/kreslenie je znovu umožnené vďaka knižnici Fabric.js a je vykonávané na priehľadný canvas element umiestnený nad obrázkom. Pri používaní tohoto módu je dôležité, aby užívateľ ohraničil ranu úplne a neboli v ohraničení nejaké prerušované časti. Keďže tento mód nie je úplne intuitívny na mobilných zariadeniach vzhľadom na ich veľkosť a odporúča sa tento mód využívať hlavne pri desktopovej verzií, tak bola pridaná možnosť resetu celého plátna, alebo iba vrátenie jendoho kroku dozadu. V prípade, že užívateľ ohraničí celú ranu, tak sa na server odošlú všetky body ohraničenia, ktoré sú na serveri rekonštruované a vykresľované pomocou OpenCV do obrázku. Zistenie plochy rany po tom ako je rana ohraničená je už rovnaké, ako v prípade poloautomatickej detekcie a to bolo popísané v kapitole Implementácia aplikačného rozhrania.

- **Nastavenie mierky a vypočítanie reálnej plochy -** Aby bolo možné zistiť veľkosť reálnej plochy rany napríklad v centimetroch, alebo inej užívateľom zadanej miery reálneho sveta a nielen veľkosť rany na obrázku v pixeloch, tak je nutné vždy určiť mierku. Toto bolo vyriešené tak, že užívateľ musí v danej snímke nastaviť úsečku na nejakú známu dĺžku. Ak na snímke sa nachádza nejaký predmet, tomuto účelu môže dobre poslúžiť obyčajné pero, a užívateľ vie rozmery tohoto pera (stačí vedieť napríklad šírku pera), tak užívateľ nastaví dĺžku tejto úsečky na šírku tohoto pera a tak sa získa mierka, podľa ktorej sa ďalej počíta reálna plocha rany. Z dĺžky úsečky/mierky sa získa koľko centimetrov štvorcových, alebo inej miery zaberá práve jeden pixel na snímke a touto hodnotou sa vynásobí počet pixelov plochy rany, viď. vzorec 5.10.

$$S_{cm} = S_{px} * (d_{cm}/d_{px})^2 \tag{5.10}$$

Úsečka je znovu vykresľovaná na plátno a jej manipulácia je možná vďaka Fabric.js. Po zvolení úsečky sú na server odoslané súradnice jej začiatočného a koncového bodu a táto úsečka je dokonca vykreslená do finálneho obrázka, aby sa aj naďalej vedelo, aký predmet, alebo aká vzdialenosť bola zvolená ako mierka.

Kapitola 6

Testovanie a vyhodnocovanie

Po naimplementovaní navrhnutého riešenia, ktoré bolo popísané v kapitole 5, nasledovalo testovanie a vyhodnocovanie, ktorému sa venuje práve táto kapitola. Kapitola začína popisom testovania serverového aplikačného riešenia pomocou testovacieho rámca Tavern, pokračuje vyhodnocovaním naimplementovanej detekcie chronických rána a na záver kapitoly sú zhrnuté ďalšie možné vylepšenia práce a je načrtnuté budúce smerovanie projektu.

6.1 Testovanie serverového aplikačného rozhrania

Testovanie aplikačného rozhrania prebiehalo už počas samotnej implementácie, kedy vždy po naprogramovaní nejakej sady koncových adries, boli napísané aj testy na overenie funkcionality týchto adries. Testovanie prebiehalo pomocou Python testovacieho rámca Tavern, ktorý je voľne dostupný pod licenciou MIT. Tavern je plugin pre Pytest, nástroj príkazovej riadky a aj Pythonova knižnica v jednom určená pre testovanie aplikačných rozhraní založených nielen na RESTe. Používa syntax založenú na YAML, ktorá je veľmi flexibilná a jednoduchá. Okrem RESTful aplikačných rozhraní dokáže testovať aj rozhrania založené na MQTT. Veľká výhoda Tavern je, že môže byť veľmi jednoducho použitý v hocijakom continous integration procese. [20]

Testy vytvorené pre Tavern sú teda definované v YAML súboroch, ktoré predstavujú danú testovaciu sadu. Táto sada je vždy definovaná svojím menom (*test_name*) a jednotlivými krokmi samotného testu (*stages*). Každý jeden krok predstavuje samostatné dotazovanie sa na server a volanie serverovej funkcie. Kroky sa skladajú z mena kroku (*name*), z požiadavky (*request*), ktorá sa vykonáva a z odpovede (*response*), ktorá sa kontroluje. V požiadavke je možné definovať všetky nastavenia, ako url adresy serveru (*url*), na ktorý sa bude posielať požiadavka, metódu, ktorá sa bude vykonávať (*method*), dáta vo formáte json, ktoré sa na server budú odosielať (*json*) a prípadne ďalšie hlavičky (*headers*), ktoré sú v požiadavke prítomné (v tomto prípade je prítomná iba hlavička *Authorization*). Správnosť odpovede, ktorá sa potom vyhodnocuje je overovaná podľa vráteného stavového kódu (*status_code*) a dát obsiahnutých v tele (*body*), poprípade iných položiek, ktoré ale v prípade tejto práce nie sú potrebné. Štruktúru takéhoto kroku testu je možné vidieť pre predstavu v kóde.

```
1 ...
2 - name: Get scale list after update
3     request:
4       url: "{env_host:s}/scale"
5       method: GET
```

```
6       headers:
7         Authorization: "{firebase_token:s}"
8     response:
9       status_code: 200
10      body:
11        - name: Scale
12          unit: px
13          value: 10
14          user: "{firebase_user:s}"
15          _id:
16            \$oid: "{scale_id:s}"
17  ...
```

Zdrojový kód 6.1: Ukážka testovacieho kroku.

Testy sú spúšťané pomocou nástroju príkazového riadku Tavern-CI. Testy tejto práce boli rozdelené do 2 súborov, a to na prípady, ktoré môžu nastať pri každodennom používaní aplikácie (*valid.tavern.yaml*) a na prípady, ktoré by mohli nastať iba v prípade nejakej neočakávanej chyby, alebo pri pokuse o hackovanie aplikácie (*invalid.tavern.yaml*). Súbor *enviroment.yaml* slúži na uchovávanie premenných, ktoré sa používajú a sú vkladané do obidvoch testov. Ide o premennú kľúča k Firebase Authentification, e-mailu a hesla k účtu testovacieho užívateľa a koreňovú adresu serveru aplikačného rozhrania. Kroky testu sa vyhodnocujú postupne a v prípade, že sa nevyhodnotí správne jeden krok, tak vyhodnocovanie ďalších krokov je zrušené a užívateľ je o tejto skutočnosti informovaný. Po otestovaní finálnej podoby aplikačného serverového rozhrania, kedy všetky testy dopadli úspešne, bolo toto rozhranie považované za stabilné a fungujúce správne.

6.2 Vyhodnocovanie detekcie chronickej rany

Vyhodnocovanie a testovanie poloautomatickej detekcie bolo vykonávané na snímkoch chronických rán, ktoré poskytla Fakultná nemocnica Brno. Ide o snímky vytvorené pomocou bežného mobilného fotoaparátu, ktorý nebol ničím špeciálny a nachádza sa takmer v každom modernom mobilnom zariadení. Získavanie snímok neprebiehalo za prítomnosti žiadnych špeciálnych predpripravených podmienok. Snímky sú rôznej kvality a zachycujú rôzne chronické rany v rôznych štádiách, od rán, ktoré sa skoro neprejavili, až po rany, ktoré už boli v pokročilom štádií a značne komplikované ako na liečbu, tak aj na samotnú detekciu. Snímky boli získavané nemocnicou približne od polovice januára 2018 do polovice mája 2018 a takto bolo získaných 40 snímok zo 14 unikátnych defektov (v rovnakom čase bol ten istý defekt zachytený niekoľkokrát). Z týchto 14 defektov sa avšak nejednalo vždy o chronické rany, ale objavila sa aj snímka stareckých škvŕn, alebo jedno približne 60 ročné materské znamienko. Unikátnych chronických rán bolo teda dokopy 12 a z toho 10 bolo použitých pre vyhodnocovanie. Všetky unikátne rany boli autorom práce anotované za pomoci externého konzultanta tejto práce.

Výskumy, ktoré sa zaoberali detekciou chronických rán, alebo príbuzných defektov a boli behom práce na tomto projekte preštudované, boli vykonávané pod určitými kontrolovanými podmienkami, ako napríklad špeciálne nasvietenie rany a podobne (štúdie [43], [30] a [35]). Detekcia a analýza iných rán zase prebiehala iba na ručne predspracovaných snímkach, napríklad bez pozadia, alebo snímkach, ktoré obsahovali iba samotnú ranu (štúdie [41] a [29]). Iné štúdie ako [44] alebo [40] sa zase zamerali iba na určitý špecifický typ chronickej rany. Všetky tieto štúdie boli overované na dátovej sade obsahujúcej od 5 v [44] do 113 snímkov v [41]. Ich presnosť detekcie za týchto, dá sa povedať zjednodušených pod-

mienok, dosahovala presnosť od 56,4% v [43] do 98% v [29]. Dá sa ale predpokladať, že takéto podmienky pri získavaní snímok pomocou mobilného zariadenia v bežnej lekárskej praxi nastať nemôžu, alebo iba veľmi ťažko. Sťažením tohoto projektu oproti spomenutým štúdiám bolo avšak to, že snímky neboli získavané behom žiadnych kontrolovaných podmienok, neboli ručne predspracované a taktiež kvôli nedostatku testovacieho materiálu bola práca zameraná na všetky typy chronických rán, nie iba na niektorých špecifických.

Samotné vyhodnocovanie a testovanie bolo vykonávané tak, že testovacie snímky boli manuálne ohraničené externým konzultantom z nemocnice pomocou špeciálne vytvorenej webovej aplikácie pre tieto účely, ktorú je možné vidieť na obrázku 6.1, a tak vznikli akési predlohy na porovnávanie a vyhodnotenie poloautomatickej detekcie. Je nutné podotknúť,

Obr. 6.1: Webová aplikácia na ohraničovanie snímok konzultantom

že chronické rany sú veľmi zložité a keď sa stretne viacero špecialistov, tak môžu mať rozdielny názor na to, čo ešte patrí do priestoru rany a čo už rana nieje. Je teda nutné pri detekcii a aj tvorbe predlohy a porovnávaní s ňou vždy rátať s určitou dávkou subjektivity. Po označení rán boli na snímkach rany potom znovu detekované pomocou poloautomatickej metódy detekcie. Získaná plocha poloautomatickou detekciou je následne porovnávaná s označenou plochou a to tak, že pixel, ktorý je v oblasti rany pri poloautomatickej detekcií a aj pri manuálnom ohraničení je správne detekovaný pixel a pixel, ktorý je naopak detekovaný iba pri poloautomatickej detekcii je nesprávne detekovaný pixel. Úspešnosť detekcie je potom daná vzťahom 6.1, kde $SuccessfulDetected$ predstavuje počet správne detekovaných pixelov a $ManualDetected$ počet pixelov v manuálne ohraničenej rane.

$$SuccessRate = SuccessfulDetected * 100/ManualDetected \tag{6.1}$$

Výsledky je možné vidieť v tabuľke 6.1, kde je vždy uvedené číslo rany, ktoré odpovedá menu súboru z priečinku *shots/detected* nachádzajúceho sa na priloženom nosiči, počet detekovaných pixelov, počet očakávaných pixelov (pixelov v manuálne ohraničenej rane), počet pixelov, ktoré boli správne detekované spolu s percentuálnou úspešnosťou a nakoniec počet pixelov, ktoré boli detekované nesprávne. Všetky výsledky detekcie v obrazovej forme je

Rana	Detekované	Očakávané	Správne detekované	Nesprávne detekované
1	6280px	13820px	6213px (44.96%)	67px
2	0px	261px	0px (0.00%)	0px
3	1108px	10296px	1041px (10.11%)	67px
4	10484px	2244px	2244px (100.00%)	8240px
5	7499px	8285px	5424px (65.47%)	2075px
6	3026px	2495px	2038px (81.68%)	988px
7	46712px	34051px	34046px (99.99%)	12666px
8	6307px	7182px	6161px (85.78%)	146px
9	23917px	2301px	2301px (100.00%)	21616px
10	5509px	6560px	5022px (76.55%)	487px

Tabuľka 6.1: Výsledky vyhodnocovania poloautomatickej detekcie

taktiež možné vidieť na priloženom nosiči v priečinku *shots/detected*. V tabuľke výsledkov je možné si všimnúť, že výsledné hodnoty boli rôzne, tak ako jednotlivé rany. Behom práce bolo navrhnuté, aby sa projekt zameral len na určitý druh rán v určitom štádií, vzhľadom na to, aké až príliš rozdielne boli jednotlivé rany na snímkach. Bohužiaľ ale behom pol roka nebolo možné získať taký dostatočný počet snímkov, aby sa bolo možné užšie zameranie. Z obrázkov je jasné, že čo rana to v podstate iný druh v inom štádiu a preto boli v rámci implementácie zvolené postupy také aké boli, teda okrem poloautomatickej detekcie algoritmom Region Growing bol pridaný aj manuálny mód ohraničenia rany. Z tabuľky je možné si všimnúť hodnoty úspešnosti detekcie, ktoré sú presne 100%. Rany, ktoré majú takúto úspešnosť boli síce detekované celé, avšak do okolia rany bol zahrnutý aj predstupeň pred samotným defektom. Takúto ranu je možné vidieť na obrázkoch 6.2 a 6.3. Naopak veľmi dobre bola detekovaná rana, ktorá bola jasne iná ako okolitá pokožka a to tak, že aj lajk by dokázal ranu ohraničiť. Ranu je možné vidieť na obrázkoch 6.4 a 6.5.

6.3 Ďalšie smerovanie projektu

Ako je možné zistiť podľa výsledkov, tak poloautomatická detekcia funguje rôzne na rôznych snímkach. Zatiaľ čo funguje na ranách, ktoré sú farebne dobre oddelené od okolitej kože, tak podstatne viac zlyháva na ranách, ktoré sú menej výrazné, alebo sú pomerne komplikované. Avšak, aby boli tieto tvrdenia potvrdené, je nutné urobiť ďalšiu a oveľa väčšiu sadu snímkov, a keďže behom 5 mesačného obdobia bolo bolo zaznamenaných 12 unikátnych defektov, typu chronická rana, tak tento proces bude nezanedbateľne časovo náročný. Z toho dôvodu nie je úplne overená funkčnosť poloautomatickej detekcie. Aplikácia však môže byť využívaná, tak, že sa bude používať poloautomatická detekcia na pomerne jednoduchších a farebne výraznejších ranách, a na zložitejších ranách bude využívaná manuálna detekcia pomocou ohraničenia rany prstom/myšou. Okrem toho, že poloautomatická detekcia bola testovaná na chronických ranách, tak bola testovaná aj na pár snímkach hematómov a jednom výraznom znamienku, ktoré nemocnica taktiež poskytla. Ako je vidieť na obrázkoch 6.6 a 6.7, tak aplikácia môže byť použitá aj na takéto prípady. Poprípade sa môže algoritmus ľahko upraviť, aby mohli byť detekované všetky tieto prípady, keďže hematómy sú v porovnaní s chronickými ranami oveľa menej rozmanité a teda viac farebne navzájom podobné.

Obr. 6.2: Rana 4 - manuálna detekcia, očakávaná

Obr. 6.3: Rana 4 - automatická detekcia.

Obr. 6.4: Rana 8 - manuálna detekcia, očakávaná

Obr. 6.5: Rana 8 - automatická detekcia

Obr. 6.6: Detekcia hematómu

Obr. 6.7: Detekcia znamienka

Kapitola 7

Záver

Cieľom diplomovej práce bolo vytvorenie aplikácie pre zariadenia s operačným systémom Android a zariadenia s operačným systémom Windows pre detekciu, lokalizáciu a určenie plochy chronických rán. Táto aplikácia má slúžiť zdravotným sestrám, doktorom a ošetrovateľom ako pomoc pri vyhodnocovaní a sledovaní chronickej rany počas trvania liečby. Aplikácia je postavená na programovacom jazyku Typescript, hybridnom aplikačnom rámci Ionic a Electron. Detekcia chronickej rany prebieha na strane servera, kde sa využíva programovací jazyk Python, pre RESTful aplikačné rozhranie aplikačný rámec Flask a pre spracovanie obrazu knižnica OpenCV. Dáta sa ukladajú do NoSQL databáze MongoDB.

Práca na začiatku začínala kapitolou 2, ktorá sa venovala teoretickému rozboru a obsahuje informácie potrebné k základnému pochopeniu problematiky chronických rán. Ďalšia kapitola 3 sa zaoberala analýzou a návrhom samotnej mobilnej aplikácie a jej serverových súčastí. Konkrétne popisovala návrh grafického užívateľského rozhrania, ale aj návrh aplikačného rozhrania REST a výber vhodných technologických prostriedkov. Hneď potom nasleduje kapitola 4, ktorá rozoberala technológie, ktoré boli potrebné pri vývoji aplikácie a jej súčastí. V poradí ďalšou a určite najvýznamnejšou kapitolou je kapitola 5, ktorá pojednávala o samotnej implementácií. Celá práca je ukončená kapitolou 6 o testovaní a vyhodnocovaní implementovaného riešenia a sú navrhnuté ďalšie kroky a smerovanie práce.

Finálna Android a Windows aplikácia, ktorá vznikla v rámci tejto diplomovej práce dokáže detekovať, lokalizovať a určiť plochu zosnímanej chronickej rany. Aplikácia detekuje tieto rany poloautomaticky výberom bodu vo vnútri rany za pomoci Region Growing algoritmu nad RYKW pravdepodobnostnou mapou, alebo v prípade zložitejších chronických rán, kedy poloautomatická detekcia výrazne zlyháva je možné použiť manuálnu detekciu pomocou ohraničenia plochy rany pomocou prsta, alebo myši. Výpočet reálnej plochy rany je potom zabezpečený pomocou užívateľom zvolenej mierky, kedy sa vždy jeden pixel plochy prevádza do nejakej miery reálneho sveta. Aplikácia je teda použiteľná do určitej miery v klinickej praxi, avšak mala by byť ešte ďalej testovaná a vylepšovaná, poprípade by sa mala užšie zamerať na niektoré špecifické typy rán v konkrétnom štádií. Aplikácia ďalej môže byť ešte rozšírená tak, aby bola schopná detekovať rôzne iné defekty kože. Takýmito príkladmi môžu byť napríklad hematómy, alebo materské znamienka.

Literatúra

[1] About JavaScript. [online]. 2017 [cit. 2017-12-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[2] Angular Framework. [online]. [cit. 2017-12-11]. Dostupné z: https://angular.io/.

[3] Apache Cordova. [online]. [cit. 2017-12-11]. Dostupné z: https://cordova.apache.org/.

[4] ArrayBuffer. [online]. [cit. 2018-03-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer.

[5] Blob. [online]. [cit. 2018-03-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Blob.

[6] Bércový vred. *Wikiskripta*, ISSN 1804-6517, [online]. [cit. 2017-11-08]. Dostupné z: http://www.wikiskripta.eu/index.php/Baz%C3%A1ln%C3%AD_ganglia.

[7] The Dependency Injection pattern. [online]. [cit. 2018-03-28]. Dostupné z: https://angular.io/guide/dependency-injection-pattern.

[8] Diabetická noha. *Wikiskripta*, ISSN 1804-6517, [online]. 2017 [cit. 2017-11-08]. Dostupné z: http://www.wikiskripta.eu/w/Diabetick%C3%A1_noha.

[9] Electron Documentation. [online]. [cit. 2017-12-11]. Dostupné z: https://electronjs.org/docs/tutorial/about.

[10] Flask – Overview. [online]. [cit. 2017-12-05]. Dostupné z: https://www.tutorialspoint.com/flask/flask_overview.htm.

[11] Hojení ran. [online]. 2017 [cit. 2017-10-29]. Dostupné z: https://www.hojeni-ran.cz/.

[12] HSV (Hue, Saturation and Value). [online]. [cit. 2018-04-04]. Dostupné z: http://www.tech-faq.com/hsv.htm.

[13] HTML - Overview. [online]. [cit. 2017-12-09]. Dostupné z: https://www.tutorialspoint.com/html/html_overview.htm.

[14] Ionic Framework. [online]. [cit. 2017-12-11]. Dostupné z: https://ionicframework.com/.

[15] Multimediálna e-učebnica. [online]. [cit. 2017-11-08]. Dostupné z: http://oschir.jfmed.uniba.sk/CCH2-5.php.

[16] @ngrx/effects. [online]. [cit. 2018-02-26]. Dostupné z: https://github.com/ngrx/platform/blob/master/docs/effects/README.md.

[17] @ngrx/store. [online]. [cit. 2018-02-26]. Dostupné z: https://github.com/ngrx/platform/blob/master/docs/store/README.md.

[18] Python - Overview. [online]. [cit. 2017-12-05]. Dostupné z: https://www.tutorialspoint.com/python/python_overview.htm.

[19] Sass - Overview. [online]. [cit. 2017-12-09]. Dostupné z: https://www.tutorialspoint.com/sass/sass_overview.htm.

[20] Tavern. [online]. 2018-04-14 [cit. 2018-04-23]. Dostupné z: https://taverntesting.github.io/.

[21] TypeScript - Overview. [online]. [cit. 2017-12-11]. Dostupné z: https://www.tutorialspoint.com/typescript/typescript_overview.htm.

[22] What is CSS? [online]. [cit. 2017-12-09]. Dostupné z: https://www.tutorialspoint.com/css/what_is_css.htm.

[23] Čo sú to hybridné mobilné aplikácie? [online]. 2016 [cit. 2017-12-11]. Dostupné z: https://robime.it/co-su-hybridne-mobilne-aplikacie/.

[24] Allamaraju, S.: *RESTful Web services cookbook*. Sebastopol,CA.: O'Reilly, první vydání, c2010, ISBN 978-0596801687.

[25] Banker, K.: *MongoDB in action*. Greenwich, CT: Manning, druhé vydání, 2011, ISBN 1935182870.

[26] Bradski, G. R.: *Learning OpenCV*. Sebastopol: O'Reilly, c2008, ISBN 978-0-596-51613-0.

[27] Ebling, G.; Ebling, F. J. G.: Encyclopædia Britannica. [online]. [cit. 2017-11-07]. Dostupné z: https://www.britannica.com/science/human-skin.

[28] Fauzi, M. F. A.; Khansa, I.; Catignani, K.; aj.: Computerized segmentation and measurement of chronic wound images. *Computers in Biology and Medicine*, ročník 60, 2015: s. 74 – 85, ISSN 0010-4825, doi:https://doi.org/10.1016/j.compbiomed.2015.02.015.

[29] Hani, A. F. M.; Arshad, L.; Malik, A. S.; aj.: Haemoglobin distribution in ulcers for healing assessment. In *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, ročník 1, June 2012, s. 362–367, doi:10.1109/ICIAS.2012.6306219.

[30] Hettiarachchi, N. D. J.; Mahindaratne, R. B. H.; Mendis, G. D. C.; aj.: Mobile based wound measurement. In *2013 IEEE Point-of-Care Healthcare Technologies (PHT)*, Jan 2013, ISSN 2377-5262, s. 298–301, doi:10.1109/PHT.2013.6461344.

[31] Hlinková, E.; Nemcová, J.; Miertová, M.: *Nehojace sa rany*. Martin: Osveta, první vydání, 2015, ISBN 978-80-8063-433-9.

[32] Hordějčuk, V.: REST. *Voho*, [online]. [cit. 2017-12-05]. Dostupné z: http://voho.eu/wiki/rest/.

[33] Hossain, M.: *CORS in action*. Shelter Island, NY: Manning, [2015], ISBN 161729182X.

[34] Jahoda, B.: Single page application. [online]. 2015 [cit. 2017-12-11]. Dostupné z: http://jecas.cz/spa.

[35] Loizou, C. P.; Kasparis, T.; Mitsi, O.; aj.: Evaluation of wound healing process based on texture analysis. In *2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE)*, Nov 2012, s. 709–714, doi:10.1109/BIBE.2012.6399754.

[36] Menon, R.: Base64 Explained. [online]. [cit. 2018-03-28]. Dostupné z: https://blogs.oracle.com/rammenon/base64-explained.

[37] Neckář, J.: Singleton. [online]. [cit. 2018-03-27]. Dostupné z: https://www.algoritmy.net/article/1326/Singleton.

[38] Pokorná, A.; Mrázová, R.: *Kompendium hojení ran pro sestry*. Praha: Grada, první vydání, 2012, ISBN 978-80-247-3371-5.

[39] Sengstacke, P.: JavaScript Transpilers. [online]. [cit. 2018-03-20]. Dostupné z: https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them.

[40] Song, B.; Sacan, A.: Automated wound identification system based on image segmentation and Artificial Neural Networks. In *2012 IEEE International Conference on Bioinformatics and Biomedicine*, Oct 2012, s. 1–4, doi:10.1109/BIBM.2012.6392633.

[41] Veredas, F.; Mesa, H.; Morente, L.: Binary Tissue Classification on Wound Images With Neural Networks and Bayesian Classifiers. *IEEE Transactions on Medical Imaging*, ročník 29, č. 2, Feb 2010: s. 410–427, ISSN 0278-0062, doi:10.1109/TMI.2009.2033595.

[42] Vilímovský, M.: Dekubity (proleženiny), jejich příčiny a léčba. [online]. [cit. 2017-11-08]. Dostupné z: https://cs.medlicker.com/848-dekubity-prolezeniny-jejich-priciny-a-lecby.

[43] Wannous, H.; Treuillet, S.; Lucas, Y.: Supervised Tissue Classification from Color Images for a Complete Wound Assessment Tool. In *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2007, ISSN 1094-687X, s. 6031–6034, doi:10.1109/IEMBS.2007.4353723.

[44] Wantanajittikul, K.; Auephanwiriyakul, S.; Theera-Umpon, N.; aj.: Automatic segmentation and degree identification in burn color images. In *The 4th 2011 Biomedical Engineering International Conference*, Jan 2011, s. 169–173, doi:10.1109/BMEiCon.2012.6172044.

[45] Švehlík, J.: Odstránenie znamienok. [online]. [cit. 2017-12-08]. Dostupné z: http://www.svehlik.sk/znamienka.html.

Príloha A

Obsah CD

Adresárová štruktúra CD

- *src* - Zdrojové súbory
  - *app* - Zdrojové súbory aplikácie
  - *api* - Zdrojové súbory aplikačného rozhrania
  - *helper* - Zdrojové súbory aplikácie na ohraničovanie rán konzultantom
- *demo* - Demonštračné súbory
  - *android* - Android aplikácia
  - *windows* - Windows aplikácia
  - *tutorial* - Video návod
- *shots* - Snímky rán
  - *anotated* - Anotované snímky
  - *detected* - Výsledky detekcie
- *text* - Text práce
  - *pdf* - Výsledný PDF dokument
  - *src* - Zdrojové súbory

Príloha B

Návod na inštaláciu

Android a Windows aplikácia je dostupná na priloženom CD a Android aplikácia dokonca aj v obchode Google play. Aplikačné rozhranie beží na autorovom virtuálnom privátnom servery http://194.182.70.49:5000/. Avšak v prípade nutnosti, kedy by bolo požadované znovu nasadiť aplikačné rozhranie na server, alebo zostaviť aplikáciu je prítomný tento návod.

B.1 Nasadenie serverového aplikačného rozhrania

Serverové aplikačné rozhranie vyžaduje pre svoj beh Python verziu 3.5.2, a MongoDB verziu 3.6.2. Pre nainštalovanie závislostí sa vyžaduje správca balíkov pip3 verzia 8.1.1. Po tom, ako cieľový stroj obsahuje všetky tieto programy, je možné spustiť skript instaler.sh, ktorý nainštaluje potrebné závislosti hlavne pomocou pip3 (skript je tvorený pre operačné systémy založené na Ubuntu, používa okrem pip3 aj apt-get). Následne je nutné spustiť MongoDB databázu pomocou príkazu *mongod*, nastaviť cestu k hlavnému súboru (*export FLASK_APP=main.py*) a spustiť server pomocou príkazu *flask run*. Server následne bude načúvať na porte 5000.

B.2 Zostavenie aplikácie

Aplikácia pre zostavenie vyžaduje Node verziu 9.4.0 a správcu balíkov NPM verziu 5.6.0. Samotný Ionic je potrebné nainštalovať vo verzií 3.20.0. Po nainštalovaní rámca Ionic je potom možné spustiť inštalovanie závislostí pomocou príkazu *npm install* v koreni adresára projektu a pridať Cordova platformy android a browser príkazmi *ionic cordova platform add android* pre Android verziu a *ionic cordova platform add browser* pre Windows desktop verziu. Pre zostavenie android aplikácie je nutné následne spustiť príkaz *ionic cordova build android –prod –release*, alebo *ionic cordova build browser –prod –release*. Desktopovú verziu je potom nutné ešte zabaliť do *asar* archívu spustením skriptu *electron/packager.sh*, ktorý vytvorí tento archív s aplikáciou pre Electron v zložke *electron-dist* a následne tento archív priložiť k samotnému Electronu.

Príloha C

Android aplikácia

Obr. C.1: play.google.com/store/apps/details?id=net.raiper34.wounddetector